1986

A computer aided design for the generation of test transactions and test databases and for the benchmarking of parallel, Multiple Backend Database Systems.

Fenton, George Patrick.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A COMPUTER AIDED DESIGN FOR THE GENERATION OF TEST TRANSACTIONS AND TEST DATABASES AND FOR THE BENCHMARKING OF PARALLEL, MULTIPLE BACKEND DATABASE SYSTEMS

by

George Patrick Fenton

June 1986

Thesis Advisor: David K. Hsiao

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): 52	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification) UNCLASSIFIED A Computer Aided Design for the Generation of Test Transactions and Test Databases and for the Benchmarking of Parallel, Multiple Backend Database Systems

12. PERSONAL AUTHOR(S): George Patrick Fenton

13a TYPE OF REPORT: Masters Thesis	13b TIME COVERED: FROM _____ TO _____	14 DATE OF REPORT (Year, Month, Day): 1986 June 20	15 PAGE COUNT: 134

16 SUPPLEMENTARY NOTATION

COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number): CAD, computer aided design, benchmark, capacity-growth, database, database management systems, multi-backend database system (Continued)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The multi-backend database system (MBDS) is a research effort conducted jointly by the Naval Postgraduate School and Ohio State University with the sponsorship of the STARS foundation. The MBDS is designed to overcome the capacity growth and performance gain problems of the traditional database systems and the single-backend database systems.

To verify the aforementioned capacity growth and performance claims, a benchmarking methodology has been formulated in the Naval Postgraduate thesis, "A Performance Measurement Methodology for Software Multiple-Backend Database Systems" by James R. Vincent. This thesis presents the implementation of the methodology in the form of a computer-aided design (CAD) system for the generation of test databases and test-transaction mixes for benchmarking MBDS.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Prof. David K. Hsiao	22b TELEPHONE (Include Area Code): 408 646-2253; 22c OFFICE SYMBOL: 52Hq

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

Block # 18 (Continued)

MBDS, performance evaluation, performance-gain, response-time reduction, response-time invariance, test databases, and test-transaction mixes.

A Computer Aided Design for the Generation of Test Transactions and Test Databases and for the Benchmarking of Parallel, Multiple Backend Database Systems

by

George P. Fenton
Major, United States Marine Corps
B.S., United States Military Academy, 1974
M.A., Central Michigan University, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1986

ABSTRACT

The multi-backend database system(MBDS) is a research effort conducted jointly by the Naval Postgraduate School and Ohio State University with the sponsorship of the STARS foundation. The MBDS is designed to overcome the capacity growth and performance gain problems of the traditional database systems and the single-backend database systems.

To verify the aforementioned capacity growth and performance claims. a benchmarking methodolgy has been formulated in the Naval Postgraduate Thesis. "A Performance Measurement Methodology for Software Multiple-Backend Database Systems" by James R. Vincent. This thesis presents the implementation of the methodology in the form of a computer-aided design (CAD) system for the generation of test databases and test-transaction mixes for benchmarking MBDS.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

This thesis is part of ongoing database systems research efforts being conducted at the Laboratory for Database Systems Research at the Naval Postgraduate School, Monterey, Ca 93943, under the direction of Dr. David K. Hsiao. This research is supported by grants from the Department of Defense STARS Program, and from the Office of Naval Research.

I would like to express my appreciation to the following people:

Dr. David K. Hsiao for his guidance, wisdom, and patience in developing this thesis.

Steve Demurjian, a PhD student in Computer Science, who is currently serving as a research assistant at the Naval Postgraduate School. Steve's keen insight, technical expertise, patience, and selflessness were continuously tapped throughout all phases of this thesis!

Andy Hunt, a fellow student in Computer Science, who continuously offered his support and expertise.

And finally, a special thanks to my wife, Vicky, and my children, Kelly, Kevin, Jessica, and Patrick, for their love, patience, and understanding over these past nine months.

I. INTRODUCTION

A. THE BACKGROUND

Information processing has a special significance in today's real world applications. Government, private, and commercial organizations are increasingly dependent on timely, accurate information. Within the computer science community, several areas of research have sprouted new, innovative approaches to satisfy the information needs of these organizations. Today, numerous variations of **database systems** have flooded the market. Database systems consist of both hardware components and software packages designed for fast, accurate, efficient, and economical information processing. These systems are better known as **database management systems** (DBMS).

1. Three Database-System Approaches

Three database-system approaches have emerged: 1) the traditional mainframe-based system, 2) the software single-backend system, and 3) the software multiple-backend system [Ref. 1:pp. 3-5]. The software multiple-backend system is designed to overcome the upgrade and performance problems experienced with both the traditional and the single-backend systems. This system is more unconventional than the first two and is a new kind of database computer. The software multiple-backend system is new because it resembles

neither the the traditional approach to database management by placing the system software in a mainframe computer such as SQL/DS in an IBM 3033, nor the more recent approach to database management by utilizing the dedicated hardware

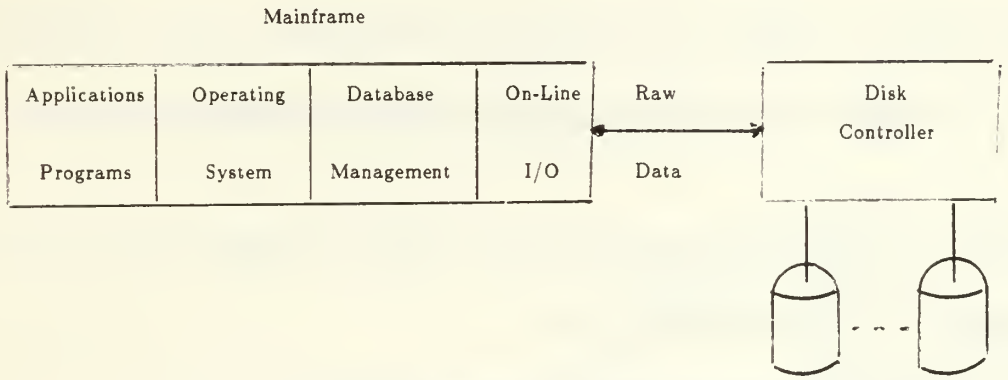


Figure 1. The Traditional Approach to Database Management

and software in a single-backend computer such as the Britton-Lee IDM 500.

In the **mainframe-based approach**, a database system is characterized as an applications program (albeit, a major one) which shares the resources of the mainframe computer with other applications programs (depicted in Figure 1). In the **single-backend approach** a database system has the exclusive control and use of the resources of the entire backend computer (depicted in Figure 2). The term **backend** connotes the 'back' of terminals or general-purpose computers, where neither the terminals nor the general-purpose computers (termed the **hosts**) provide the database management services. Instead, the database

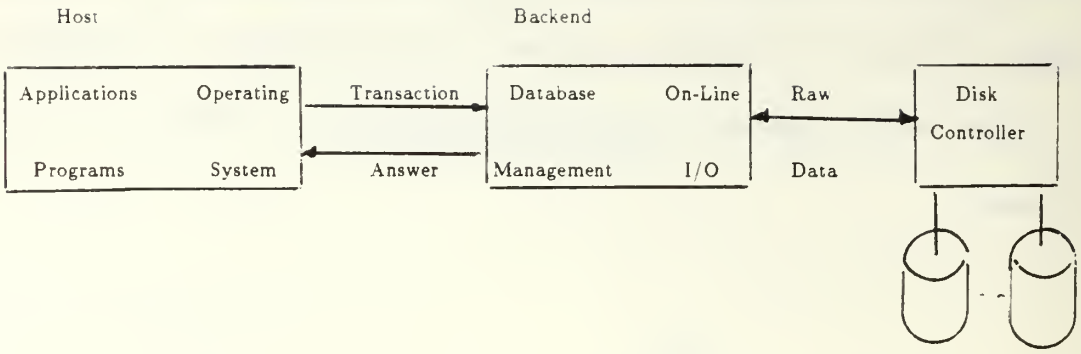


Figure 2. The Software Single-Backend Approach to Database Management

management services are provided by the backend computer to the user or user programs (**transactions**) via the host.

2. Software Multiple-Backend Database Systems

The new kind of database computers (depicted in Figure 3) is of the **multiple-backend approach** where the database system is not mainframe-based and each database system consists of at least one controller and two or more backends with their disk systems interconnected by a network. The controller software is mainly dedicated to 1) the communication with the hosts and the terminals, 2) the scheduling and control of transaction executions by the backends, and 3) the routing of the responses coming from the backends back to the user. The backend software in each of the multiple backends is identical and is responsible for carrying out the primary database operations such as the retrieve, insert, delete, and update for the transactions. Examples of the multiple-backend approach to database management are the commercial Teradata

DBC/1012 system, and the Naval Postgraduate School's experimental multi-backend database system (MDBS) [Ref. 1:pp. 5-6].

Unlike the mainframe-based and single-backend approaches, the multiple-backend approach emphasizes great-capacity and high-performance database management. Furthermore, it attempts to relate the capacity growth and performance gains to the number of backends in the system. In other words, when new backends and their disk systems are added to a multiple-backend

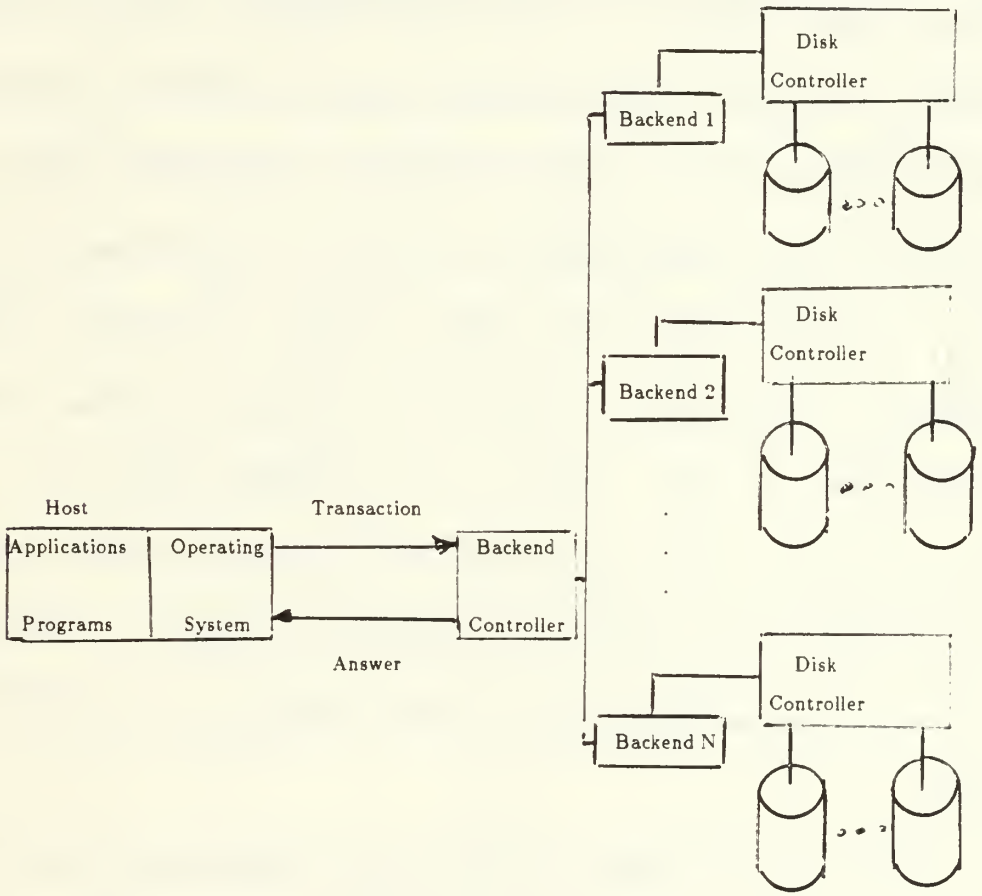


Figure 3. The Software Multiple-Backend Approach to Database Management

database computer, an increase in both the capacity and the performance would likely be produced. The MDBS system is expandable in terms of tens of backends and associated disk systems, and the DBC/1012 is expandable in terms of hundreds of backends and disks. Certainly, in the former system, the capacity growth and performance can be measured in tens and in the latter in hundreds.

B. PERFORMANCE AND CAPACITY-GROWTH CLAIMS

To overcome the performance problems and upgrade issues of the traditional mainframe-based approach and of the conventional software single-backend approach, the software multiple-backend approach provides performance gains through specialization of the database operations on dedicated, multiple backends. The system utilizes multiple backends connected in a parallel fashion in order to achieve performance gains and capacity growth. The backend controller is responsible for supervising the execution of database transactions and for the interfacing with the hosts and users. The backends perform the database operations with the database stored and evenly distributed across the disk systems of the backends. The controller and backends are connected by a communications bus. Users access the system either by way of the hosts or through the controller directly.

The two goals of the software multi-backend database system are of course to overcome the performance problems and upgrade issues of the traditional mainframe-based or the conventional software single-backend database systems.

First, by increasing the number of backends, while the size of the database and the size of the responses to the transactions remain constant, the database system is to produce a reciprocal decrease in the response times of the user transactions. Second, by increasing the number of backends proportionally to the increase of transaction responses, the database system is to produce invariant response times for the user transactions.

The first goal allows the multiplicity of the backends of the database system to be directly related to the **performance gains** of the database system in terms of the **response-time reduction**. Response-time reduction means the amount of reduction in the response time of a request, when the request is processed in a system with several backends as opposed to processing the same transaction in a one backend system, while using the same test-database set[Ref. 1:p. 24]. The second goal enables the multiplicity of the backends of the system to be directly related to the **capacity growth** of the system in terms of **response-time invariance**. Response-time invariance means the amount of change in the response time of a request, when the request is processed in a one backend system with a **response set** of x records, as opposed to processing the same transaction in a system with m backends with a response set of mx records [Ref. 1:p. 24]. Since the size of the response set for a request is determined by the size of the database (i.e., large databases generate more responses for the same request), the definition of response-time invariance can be restated as the amount of change in the response time of a request, when the request is processed in a one backend

system with a database size of x records, as opposed to processing the same transaction in a system with m backends with a database size of mx records. (Response set means the set of responses returned by the backends(s) to the user as a result of processing a transaction.)

C. THE BENCHMARK OVERVIEW

To verify the aforementioned performance and growth-capacity claims, Vincent has formulated a benchmarking methodology for software multiple-backend database systems. Vincent's work [Ref. 2] provides a logical continuation of two prior projects aimed at developing a comprehensive performance measurement methodology. Kovalchik [Ref. 3] has developed a set of performance measurement tools for conducting system testing. Tekampe and Watson [Ref. 4] have developed a performance measurement methodology for database systems to provide both external and internal performance measurements by embedding timing checkpoints in the MBDS software to provide the required measurements. The information provided by the external checkpoints provides a measure of the response time.

Vincent's methodology articulates numerous parameters to insure database-independence and application-independence when generating the benchmarking information. Thus, the design of the benchmarking methodology is complicated. There is the need:

1) to have test databases which can be used for testing backends of varying numbers, for deriving partitions (clusters) of a database, and for placing the partitions on parallel stores;

2) to have test-transaction mixes which can be used for measuring primary database operations in terms of their response times, for verifying the response-time reductions due to additions of backends and the redistribution of the same database, for clarifying the response-time invariance on account of various growths in database capacity with corresponding additions of backends and backend stores;

3) to have systematic ways to generate the test databases and the test-transaction mixes, to conduct the tests, to collect the test results, to interpret the results and to verify the results against established measures (benchmarks).

The major portions of the design of the benchmarking methodology consist of the articulation and establishment of the measurement criteria and measures, the design, interpretation, and generation of the test databases, the test-transaction mixes, the test procedures, and the test configurations.

The focus of this thesis then is the design and implementation of a computer-aided design (CAD) system for the generation of test transactions and test databases that may be used for the benchmarking of parallel, multiple-backend database systems. One of the most salient features of the CAD system is to reduce the amount of user input. The user needs only to input three essential elements of information:

- the number of backends in the system to be tested,
- the amount of disk storage per backend, and
- the size of the data transfer from the secondary storage (disk) to the primary storage (main memory).

Once the user has provided these values, the CAD system automatically generates the test databases and the test transaction mixes. Furthermore, it yields a detailed report that guides the database evaluator through the testing process. The CAD system is also very generic in form, and is able to generate test database and transaction mixes for any combination of the input test variables.

The CAD system described in this thesis generates the test databases, the test transaction mixes, and an evaluator's guide for benchmarking parallel, multiple backend database systems. This CAD system is a first version; the second version will be integrated with MBDS, allowing the testing process to be controlled and managed by the CAD system. The third version will add components to collect statistics (e.g., response times) for the different tests and calculate statistics (i.e., mean and standard deviation of tests, response-time reductions and response-time invariances) that measure the performance of MBDS.

D. ORGANIZATION OF THE THESIS

The thesis is organized into three chapters in addition to this introduction. Chapter II provides a general overview of the CAD system structure. Chapter III presents the detailed design features and focuses on the major components of the CAD system: the generated test databases and transaction mixes, and the framework for the detailed user's guide (evaluator's report). Chapter IV presents a summary and the conclusions of the thesis, and offers future work in

performance evaluation of software multi-backend database systems in general, and MBDS in particular.

II. GENERAL OVERVIEW

Chapter II presents a three-part overview of the general structure of the CAD system for generating test-databases and test-transaction mixes. Part one describes the essence of Vincent's methodology [Ref. 2]. Part two briefly describes the prototype multi-backend database system under development at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, California. Finally, part three introduces the framework of the CAD system.

A. TEST-DATABASE AND TEST-TRANSACTION DESIGN FACTORS

The database design factors presented in Vincent's thesis [Ref. 2:pp. 29-48] are the backbone not only of his methodology, but of the CAD system as well. The generated test-database sets and the generated test-transaction mixes are the two major components of his performance measurement. Essentially, Vincent's methodology describes how to create these test-database sets and test-transaction mixes. The creation of the database sets and the transaction mixes is driven by three critical elements of information solicited from the user.

- the number of backends in the system to be tested,
- the amount of disk storage per backend, and
- the size of the data transfer from the secondary storage to the primary storage.

The database design factors are the system configuration considerations, the database size considerations, and the test-transaction mix considerations.

1. System Configuration Considerations

For a given test database, two sets of configurations must be generated, a set for the measurement of the response-time reduction, and a set for the measurement of the response-time invariance. The number of configurations within each set is determined by the number of backends of the system to be tested. A configuration may range from 1 backend to M backends. Consider a system with M backends and N number of bytes in the database. The database of N bytes must be evenly distributed over the backends. Depending on the configuration being used, the database must be evenly distributed to 1, 2, 3, ... or M backends. To determine a database size which permits an equal distribution of data to each backend in the system, the **least common multiple** (**lcm**) must be calculated for the possible system configurations of 1, 2, 3, ... or M backends. The lcm is used in another calculation, the **database-size multiple**, or **dbm**. The dbm is the ultimate factor used to determine the correct database size which allows for equal distribution of the database across M backends. The dbm is discussed later.

The total number of configurations for a given test database is determined by the equation:

$$\text{Number of configurations} = M + (M - 1) = 2M - 1$$

where M is the total number of backends to be tested in the system.

To verify the performance-gain claim, the database must be distributed evenly across 1, 2, 3, . . . , M backends, with each distribution comprising a configuration. For example, given a system with two backends, two configurations are need to test the response-time reduction claim. The first configuration has the entire database on one backend, and the second configuration has the database **split** evenly between the two backends. Thus for the response-time reduction calculations of a M-backend system, M configurations are needed.

To verify the growth-capacity claim, the database size must increase by a factor of 2, 3, . . . , M , with each configuration reflecting the incremental increase in size up to M. Again, given a system with two backends, two configurations are needed to test the response-time invariance claim. The first configuration has the entire database on one backend. The second configuration then **doubles** the size of the database and distributes it evenly over two backends. Thus for the response-time invariance calculations of a M-backend system, M configurations are also needed. Tables 1 and 2 reflect the database allocation with respect to the number of configurations. Notice however, that the first configuration for the response-time reduction set and the first configuration for the response-time invariance set are the same configuration. Eliminating the creation of the duplicate configuration, the total number of test configurations is equal to (M + M -1) or 2M - 1. Tables 3-5 summarize this information for systems configured with a maximum of two, three, and four backends, respectively.

TABLE 1. DATABASE ALLOCATION WITH RESPECT TO RESPONSE TIME REDUCTION

Configuration Number	Maximum Number of Backends to be used:	Portion of Database Allocated per Backend:
1	1	N
2	2	N/2
3	3	N/3
4	4	N/4
. . .	. . .	. . .
M	M	N/M

KEY. M = number of backends
N = total number of bytes in database

TABLE 2. DATABASE ALLOCATION WITH RESPECT TO RESPONSE-TIME INVARIANCE

Configuration Number	Maximum Number of Backends	Total Database Size in Mbytes.
1	1	N
2	2	N*2
3	3	N*3
4	4	N*4
. . .	. . .	. . .
M	M	N*M

KEY. M = number of backends.
N = total number of bytes in database.

2. Database-size Considerations

To adequately measure the performance characteristics of a software multiple-backend system, **three** different database sizes are preferred [Ref.2: p.33], all of which are a multiple of the base (original) size N. One size represents a small database (N/4), another size represents an intermediate size database (N/2), and the final size is the base database size N. The base (original) database

size N is determined by the database-size multiple. (dbm). The dbm is calculated as follows:

$$dbm = lcm\{1.2.3,....M\} \times 32 \times record\text{-}size$$

The lcm (least common multiple) is determined using the maximum number of

TABLE 3. TEST CONFIGURATIONS WITH TWO BACKENDS.

Configuration Number:	Number of Backends:	Mbytes per Backend:	Total database Size in Mbytes
1	1	N	N
2	2	N/2	N
3	2	N	2N

Note:
Configuration's {1,2} are required to verify the response-time reduction claim
Configuration's {1,3} are required to verify the response-time invariance claim

TABLE 4. TEST CONFIGURATIONS WITH THREE BACKENDS.

Configuration Number:	Number of Backends:	Mbytes per Backend:	Total database Size in Mbytes
1	1	N	N
2	2	N/2	N
3	3	N/3	N
4	2	N	2N
5	3	N	3N

Note:
Configuration's {1.2.3} are required to verify the response-time reduction claim
Configuration's {1.4,5} are required to verify the response-time invariance claim

TABLE 5. TEST CONFIGURATIONS WITH FOUR BACKENDS.

Configuration Number:	Number of Backends:	Mbytes per Backend:	Total database Size in Mbytes
1	1	N	N
2	2	N/2	N
3	3	N/3	N
4	4	N/4	N
5	2	N	2N
6	3	N	3N
7	4	N	4N

Note:
Configuration's {1,2,3,4} are required to verify the response-time reduction claim
Configuration's {1,5,6,7} are required to verify the response-time invariance claim

backends, M, in the system. The record-size value is hardware specific because it depends on the size of the unit of data management of the particular system's architecture [Ref. 5:pp. 16-17]. The size of the unit of data management is the size of the data transfer from the secondary storage to the primary storage. This element of information is one of three elements of information solicited from the user. Strawser [Ref. 5:pp. 16-17] offers a scheme for dividing this disk track size into **four** record sizes. The largest of the four record sizes is used in the dbm calculation. The largest record size must also be divisible by each of the three smaller record sizes to simplify the database sizing process. The CAD system implements Strawser's scheme by having the four record sizes as fixed multiples of one another.

Vincent's methodology explains the requirement for the database-multiple (dbm) to be a multiple of 32 [Ref. 2:p. 44]. Basically, the number 32 is necessary because:

- the database is to be equally divided among the four record sizes. i.e., divisible by 4.
- the decision to have the smallest database (N/4) to be one fourth the size the base database size N.i.e., divisible by another 4. and
- a parameter relating to the MBDS storage mechanism,(i.e., each MBDS cluster must hold an even number of records) i.e., divisible by 2.

Thus. 4 x 4 x 2 = 32.

The final criteria for how large the database size may be is of course dependent on the available disk storage of the type of backend to be used in the system. Obviously, for testing the database size cannot be larger than the available disk storage of the single backend. The database-multiple determines exactly how much of the available disk storage is used to accommodate the largest database size. A trivial example consists of

dbm = 3 units.

available disk storage = 10 units. and

largest database size = 9 units.

Note that the dbm is able to be folded into the available disk storage 3 times. thus allowing for a database size of 9. If the dbm was 4 units, then the largest database size would be 8. A more concrete example follows. A system is defined to have the following:

available disk storage:	300	Megabytes
disk track size:	4000	bytes
number of backends:	3	

Strawser's scheme (based on the disk track size) for selecting record sizes is:

large record size:	2000	bytes
med-lg record size:	1000	bytes
medium record size:	400	bytes
small record size:	200	bytes

The calculations are:

lcm = 6 { 1, 2, and 3 backends}
dbm = lcm x 32 x large record size = 6 x 32 x 2000 = 384,000 bytes
N in Mbytes = 299.904 where 384,000 can be folded into 300 Mbytes
781 complete times

Table 6 reflects the database size calculations when the number of backends is allowed to increase. Note the dbm for 11 backends. The dbm itself exceeds the available disk storage of 300 Mbytes! Thus, the maximum number of backends allowable in the system is 10. The database design factors may be summarized as follows:

1) three database sizes that best measure the performance claim are N, N/2, and N/4.

2) the number of backends in the system is used to calculate the least common denominator, lcm.

3) the four record sizes are calculated using Strawser's scheme. These calculations are dependent on the disk track size. The largest record size is used to help calculate the database-multiple.

4) the database-multiple is calculated. The dbm is dependent on the lcm and the largest record size.

TABLE 6. MBDS DATABASE SIZE CALCULATIONS.

M	LCM of {1.....M}	DBM in Bytes	N in Mbytes	N/2 in Mbytes	N/4 in Mbytes
2	2	128,000	299.904	149.952	74.976
3	6	384,000	299.904	149.952	74.976
4	12	768,000	299.520	149.760	74.880
5	60	3,840,000	299.520	149.760	74.880
6	60	3,840,000	299.520	149.760	74.880
7	420	26,880,000	295.680	147.840	73.920
8	840	53,760,000	268.800	134.400	67.200
9	2,520	161,280,000	161.280	80.640	40.320
10	2,520	161,280,000	161.280	80.640	40.320
11	27,720	1,774,080,000	1,774.080	887.040	443.520

where:
M = maximum number of backends in the database.
LCM = Least Common Multiple.
DBM = {LCM{1.....M} * 32 * rec_size) for rec_size = 2000-bytes.
N = Size in Mbytes of large test database.
N/2 = Size in Mbytes of medium size test database.
N/4 = Size in Mbytes of small test database.

Assumption: Largest database allowable is 300 Mbytes.

5) the values of N, N/2, and N/4 are calculated based on the dbm and the available disk storage.

Vincent's methodology discusses formatting the databases using one of two options: (1) the use of only one record type per database, or (2) the inclusion of all four record sizes in a single database. Because the CAD system is being designed to specifically test the Naval Postgraduate School's experimental software multiple-backend system, option (2) is implemented.

3. Test-Transaction Mix Considerations

To demonstrate the response-time invariance of software multiple-backend systems, the CAD system must ensure that any increase in the number of backends in the system is accompanied by a proportional increase in the size of

the database, and in the size of the response set returned by the test-transaction mix. The selection of the test-transaction mix which permits the database size to increase in the same proportion as the increase in the response set is much more complex. The selection requires a complete understanding of the characteristics and features of the data model and data manipulation language [Ref. 2:p. 48]. Vincent has devised a methodology that creates the test-record organization, a test-database structure, and a test-transaction mix set which enables the system evaluator to use the same organization, structure, and mix for all system configurations without modification!

Vincent also cites the work of Hawthorn and Stonebreaker [Ref. 6] which suggests the use of three sets of test transactions to test the overall performance of MBDS. One set consists of **overhead-intensive queries**, the second set consists of **data-intensive queries**, and the third set consists of **multi-relation queries**. Vincent ensures that all of these factors are appropriately considered when selecting transactions to verify the performance-gain and capacity-growth claims.

4. System-Dependent Factors

All of the design factors mentioned thus far have been independent of the system with the exception of the record-size selection. Vincent's methodology has shown that the design for the database set satisfies the requirements for accommodating all required test-system configurations: Vincent has also demonstrated that the transaction-mix generation is well conceived and

appropriately encompasses the requisite considerations for verification of the performance-gain and capacity-growth claims. However, there are two considerations that are system specific, the data model and record composition. The data model used by the database system is directly related to the system's data management strategy, to include such factors as the directory structure and record distribution mechanism. The record composition may rely on specific system idioms and constraints, such as limits on field width, or on the number of fields within a record, etc.

B. THE MULTIPLE-BACKEND DATABASE SYSTEM (MBDS)

In the Laboratory for Database Systems Research, a prototyped software multi-backend database system, known as MBDS, has the same architectural organization depicted in figure 3 for the software multiple-backend approach to database management. One minicomputer or microcomputer functions as the controller, while one or more microcomputers and their disk systems serve as backends. Both the controller and the backends are interconnected by a broadcast bus. Together, the controller, the broadcast bus, and the backends comprise a database system which is specifically designed to overcome the the performance and capacity-growth problems experienced by traditional mainframe-based and conventional software single-backend database systems. The data in the MBDS system is distributed across the disk systems of each backend computer. Consequently, a user transaction may be executed

simultaneously by all backends. The initial design and and analysis of MBDS is presented in Reference 7 and Reference 8.

The MBDS hardware features a minicomputer (VAX-11-750) and 8 microcomputers (Integrated Solutions Incorporated workstations). All computer systems utilize the 4.2 BSD Unix Operating System. The VAX 11-750 serves as the controller, while the eight ISI workstations function as backends. The ISI workstation is based on the Motorola 68020 CPU, which features 16-Mbytes of virtual memory space per process. Each backend has one dedicated Control Data Corporation Winchester-type disk drive with a maximum formatted capacity of 500-Mbytes per drive. The MBDS configuration uses Ethernet as the broadcast bus among the controlled backends.

Recall there are two considerations that are system specific, the data model and record composition. The remainder of this section is devoted to the MBDS attribute-based data model and the attribute-based data language.

1. The Attribute-Based Data Model

MBDS is based on the attribute-based data model first proposed in Reference 9. In the attribute-based data model, data is considered in the following constructs: database, file, record, attribute-value pair, keyword, attribute-value range, directory keyword, non-directory keyword, directory, record body, keyword predicate, and query. Informally, a **database** consists of a collection of files. Each **file** contains a group of records which are characterized by a unique set of keywords. A **record** is composed of two parts. The first part is a collection of

attribute-value pairs or **keywords**. An attribute-value pair is a member of the Cartesian product of the attribute name and the value domain of the attribute. As an example, <POPULATION, 25000> is an attribute-value pair having 25000 as the value for the population attribute. A record contains at most one attribute-value pair for each attribute defined in the database. Certain attribute-value pairs of a record (or a file) are called the **directory keywords** of the record (file), because either the attribute-value pairs or their attribute-value ranges are kept in a **directory** for identifying the records (files). Those attribute-value pairs which are not kept in the directory are called **non-directory keywords**. The rest of the record is textual information, which is referred to as the **record body**. An example of a record is shown below.

(<FILE, USCensus>, <CITY, Monterey>, <POPULATION, 25000>,
{ Temperate climate })

The angle brackets, <,>, enclose an attribute-value pair, i.e., keyword. The curly brackets, {,}, include the record body. The first attribute-value pair of all records of a file, by convention, is the same. In particular, the attribute is FILE and the value is the file name. A record is enclosed in the parenthesis. For example, the above sample record is from the USCensus file.

The records of the database may be identified by keyword predicates. A **keyword predicate** is a 3-tuple consisting of a directory attribute, a relational operator (=, !=, >, <, ≥, ≤), and an attribute value, e.g., POPULATION ≥ 20000 is a keyword predicate. More specifically, it is a greater-than-or-equal-to

predicate. Combining keyword predicates in disjunctive normal form characterizes a **query** of the database. The query

(FILE = USCensus and CITY = Monterey) or
(FILE = USCensus and CITY = San Jose)

will be satisfied by all records of the USCensus file with the CITY of either Monterey or San Jose. For clarity, we also employ parentheses for bracketing conjunctions in a query.

2. The Attribute-Based Data Language (ABDL)

The attribute-based data language supports the five primary database operations, INSERT, DELETE, UPDATE, RETRIEVE, and RETRIEVE-COMMON. A **request** in the ABDL is a primary operation with a qualification. A **qualification** is used to specify the part of the database that is to be operated on. Two or more requests may be grouped together to form a **transaction**. Now, let us illustrate the five types of requests and forgo their formal specifications.

The INSERT request is used to insert a new record into the database. The qualification of an INSERT request is a list of keywords with or without a record body being inserted. In the following example, an INSERT request that

INSERT (<FILE, USCensus>, <CITY, Cumberland>, <POPULATION, 40000>)

will insert a record without a record body into the USCensus file for the city Cumberland with a population of 40,000.

A DELETE request is used to remove one or more records from the database. The qualification of a DELETE request is a query. The following example,

DELETE ((FILE = USCensus) and (POPULATION > 100000))

is a request that will delete all records whose population is greater than 100,000 in the USCensus file.

An UPDATE request is used to modify records of the database. The qualification of an UPDATE request consists of two parts, the query and the modifier. The **query** specifies which records of the database are to be modified. The **modifier** specifies how the records being modified are to be updated. The following example,

UPDATE (FILE = USCensus) (POPULATION = POPULATION + 5000)

is an UPDATE request that will modify all records of the USCensus file by increasing all populations by 5,000. In this example, (FILE = USCensus) is the query and (POPULATION = POPULATION + 5000) is the modifier.

The RETRIEVE request is used to retrieve records of the database. The qualification of a retrieve request consists of a query, a target-list, and a by-clause. The query specifies which records are to be retrieved. The target-list consists of a list of output attributes. It may also consist of an aggregate operation, i. e., AVG, COUNT, SUM, MIN, MAX, on one or more output attribute values. The optional by-clause may be used to group records when an aggregate operation is

optional by-clause may be used to group records when an aggregate operation is specified. The RETRIEVE request.

RETRIEVE ((FILE = USCensus) and (POPULATION ≥ 50000)) (CITY, POPULATION)

will retrieve the city names and populations of all records in the USCensus file whose populations are greater than or equal to 50,000. ((FILE = USCensus) and (POPULATION ≥ 50,000)) is the query and (POPULATION, CITY) is the target-list. There is no use of the by-clause or aggregation in this example.

Lastly, the RETRIEVE-COMMON request is used to merge two files by common attribute-values. Logically, the RETRIEVE-COMMON request can be considered as a transaction of two retrieve requests that are processed serially in the following general form.

RETRIEVE (query-1) (target-list-1)
COMMON (attribute-1, attribute-2)
RETRIEVE (query-2) (target-list-2)

The common attributes are attribute-1 (associated with the first retrieve request) and attribute-2 (associated with the second retrieve request). In the following example, the RETRIEVE-COMMON request

RETRIEVE ((FILE = CanadaCensus) and (POPULATION ≥ 100000)) (CITY)
COMMON (POPULATION, POPULATION)
RETRIEVE ((FILE = USCensus) and (POPULATION ≥ 100000)) (CITY)

will find all records in the CanadaCensus file with population greater than 100,000, find all records in the USCensus file with population greater than 100,000, identify records of respective files whose population figures are common.

and return the two city names whose cities have the same population figures. ABDL provides five seemingly simple database operations, which are nevertheless capable of supporting complex and comprehensive transactions.

C. THE CAD FRAMEWORK

The CAD system is being implemented in a number of versions. The first version, described in this thesis, generates the test-databases, the test transaction mixes, and an evaluator's guide, but is not integrated with MBDS. The databases, the mixes, and the guide are in fact the major components of the CAD system. The framework of this first version is to create a "TEST" directory of files for the system evaluator to manually input to MBDS. The files that are created by the CAD system and passed to the TEST directory represent both the generated test databases and the generated transaction mixes. The system evaluator is assisted in his testing by the guide provided by the CAD system.

The framework of the TEST directory for a MBDS configured with up to 3 backends is represented in Figure 4. Recalling that a 3-backend system requires a total of 5 configurations per database, i.e.,

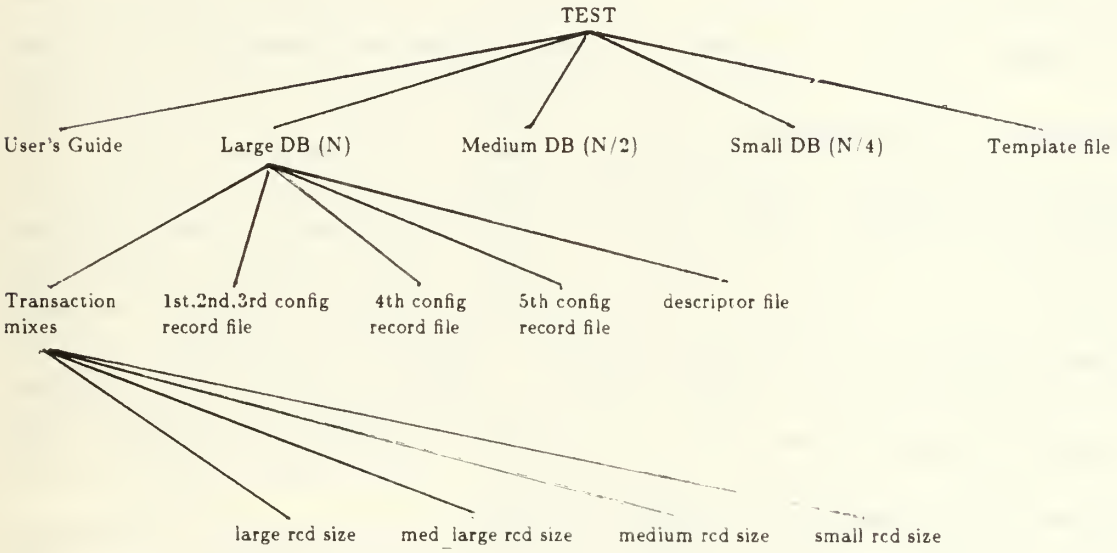
for M = 3, the number of configurations = 2M - 1 = 2(3) - 1 = 5.

Each database (small (N/4), medium (N/2), and large (N)) has its own set of configuration files as well as it own transaction-mix files. The configuration files are made up of record files, and are in fact the generated database. The transaction-mix file is used to test the performance and growth-capacity claims for

the specific database: the transaction-mix file consists of four sets of files. each set representing queries geared specifically to the record sizes CAD system generates and passes to the TEST directory a template file common to all three generated databases. The CAD system also generates a unique descriptor file for each database, small , medium and large. The descriptor file contains indexing information for the database.

TEST Directory
(a 3-backend MBDS system)

/work/username/TEST



NOTE: Each transaction mix consists of 7 sets of queries totaling 30 queries/mix/record size

The 1st,2nd.and 3rd configurations comprise a single record file which is the response-time reduction file

The 4th and 5th configuration record files constitute the response-time invariance files

Figure 4 The TEST Directory

1. Characteristics and Notions of the CAD System

The CAD system is designed to receive a minimal amount of input from the user. Upon receiving 3 essential elements of information from the user, the CAD system is to perform the following actions:

- to make a number of calculations that affect the generation of the database sets and the transaction mixes,
- to create the template file, the 3 descriptor files (one for each database size N, N/2, N/4), the three response-time reduction record files (again, one for each database size), and the requisite number of response-time invariance record files,
- and to produce an evaluator's guide as a very important by-product.

The evaluator's guide is a report that characterizes the generation of the test databases and transaction mixes. Also, the guide is the user's tool to place the generated CAD files into MBDS for testing. The most powerful characteristic of the CAD system is that just about every separate entity within the CAD system is in some form or multiple of another entity. Within the CAD system there are two important concepts that serve as the nucleus for aiding in the creation of **all** of the files that are placed in the TEST directory.

The first concept involves the creation of a number of factors that serve as multipliers for entities. For example, three database sets are generated by the CAD system with two sets being multiples of the original, e.g., N, N/2, and N/4. The first concept is to create a database factor to represent the relationships (in terms of multiples) among the database sizes. In particular, a data construct

named **dbase_factor** represents the ratios of the database sizes. In our implementation, dbase_factor is an array of three integers, 1,2 and 4. The utility of this notion is that the CAD system merely creates one database and does so with the dbase_factor set to "1". The other two databases are generated by simply accessing the appropriate dbase_factor from the array. We also apply the factor concept to represent relationships among database record sizes. Recall that Stawser's scheme for the record-size selection has 3 of the record sizes as multiples of the 4th record size. A record factor is created to represent the relationships between the record sizes. Again, in particular, a data construct named **record_factor** represents the ratios of the record sizes. In our implementation, record_factor is an array of four integers, 1, 2, 5 and 10.

The second concept is a table of numbers that reflects the distribution of records across the backends of MBDS . Recall that in MBDS, a cluster-based database placement algorithm is used to distribute clusters of records across the backends, and that this algorithm is tightly coupled to the attribute-base data model. Therefore, we develop a method by which we can successfully model the algorithm to insure an even distribution of database records of a cluster across the backends. Hence, we instantiate a data structure that can be used to model the concept. The name given to this data structure is the **base_record_&_block_distribution_table**. The numbers placed in this "base" table result from a few simple calculations dependent on the user's input. The numbers reflect the upper bounds on the number of records, blocks, and

clusters formed and distributed across the backends of MBDS. The values in the table are used to generate the appropriate number of records needed for a pre-calculated database size. They are also used to determine the range values of the descriptor files. and they serve to determine values in the transaction mixes as well.

The next chapter presents these two concepts as the building blocks for two of the major components of the CAD system. the generated test database sets and the generated test-transaction mixes.

III. THE MAJOR COMPONENTS OF THE CAD SYSTEM

The major objective of the CAD system is to generate test database sets and corresponding transaction mixes for the purpose of benchmarking parallel, multiple-backend database systems. In meeting this objective, the first version of the CAD system is comprised of three major components,

- the Test Database Generation Component,
- the Test Transaction Mix Generation Component, and
- the Evaluator's Report Component.

Before the CAD system is able to actively create any of the three major components, the CAD system initializes a number of data constructs, solicits the user for the three essential elements of information, and performs a number of initial calculations. Once the aforementioned procedure has been completed, the CAD system then generates the test-databases, followed by the generation of the test-transaction mixes. The evaluator's report is comprised of a number of text files interleaved with a number of empirical data tables generated simultaneously with both the test-databases and the test-transaction mixes. This chapter presents an in-depth study of the high-level organization of the CAD system, the components of the test-databases and test-transaction mixes, and introduces the framework of the report component.

A. HIGH-LEVEL ORGANIZATION OF THE CAD SYSTEM

For discussion the high-level organization of the CAD system is presented as three areas: (1) the characteristics of CAD, (2) the files created by CAD, and (3) the high-level program structure of CAD. The characteristics of the CAD describe the mechanisms by which the values found within the files are determined. The created files are in fact the generated test databases and the transaction mixes, and comprise a part of the evaluator's report. The high-level program structure provides an overview of the CAD system itself.

1. Characteristics of the CAD System

Chapter II introduced the two concepts that serve as the nucleus for creating **all** of the files placed in the TEST directory. The first concept involved the creation of a number of factors that serve as multipliers for entities found within the CAD system. The second concept involved the creation of a base table that reflects the distribution of records across the backends of MBDS. This section presents these two concepts as the building blocks for the generation of all files created by the CAD system.

a. The Creation of the CAD Factors

The purpose of the CAD system is to generate the template file, the descriptor files, the record files, and the transaction-mix files. A detailed discussion of these files is presented later. The first three sets of files are indicative of the attribute-based data model. The transaction-mix files are generated for benchmarking purposes. The values placed in the template files,

descriptor files, the record files, and the transaction-mix files are characteristically dependent upon

- the size of the database (small, medium, or large), and
- the record class (large, medium-large, medium, or small).

Vincent's methodology [Ref. 2] defines the ratios among the database sizes, and Strawser [Ref. 5] defines the ratios among the record classes. To represent these two sets of ratios, two data constructs are introduced. The data construct **dbase_factor** represents the ratios of the database sizes. In our implementation, dbase_factor is an array of three integers, 1, 2, and 4. The data construct **record_factor** represents the ratios of the record classes. In our implementation, record_factor is an array of four integers, 1, 2, 5, and 10. Both factors are used to generate tables of data found in the user's guide, to perform the calculations necessary to create the base-record-and-block-distribution table, and to perform the calculations for the values of the transaction mixes.

b. The Base-Record-and-Block-Distribution Table

The base-record-and-block-distribution table, hereby referred to as the **base table**, represents the records, blocks, and clusters formed and distributed across the backends of MBDS. Vincent's methodology [Ref. 2:p. 68] defines **nine** cluster categories, with each cluster category representing a cluster containing from 2 to 10 blocks of records per cluster. Thus, the base table reflects the distribution of records, blocks, and clusters across 9 categories. Table 7 depicts the layout of the base table.

TABLE 7. BASE RECORD AND BLOCK DISTRIBUTION TABLE. (BASE TABLE)

Record Size in Bytes	Number of Blocks per Cluster	Number of Records per Cluster	Total Number of Clusters	Total Number of Records	Total Number of Blocks	Number of Blocks per Backend
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					

The values for the matrix are dependent on a few initial calculations which in turn are dependent on the user's input. The nature of these calculations is discussed later. However, it is appropriate to note which columns are correlated to dbase_factor and record_factor.

column 1: record_factor

column 2: remains constant

column 3: record_factor

column 4: dbase_factor

column 5: record_factor and dbase_factor

column 6: dbase_factor

column 7: dbase_factor, number of backends, and configuration number

2. CAD-Generated Files

This section introduces the files created by the CAD system for each of the three major components. The Database Component has three types of files generated to represent the three databases:

- the **Template file**- one template file common to all three databases;
- the **Descriptor file**- three descriptor files, one per database; and

- the **Record file**- a number of record files (dependent on the number of backends in the system) representing both the response-time-reduction and the response-time-invariance configurations.

The Transaction-Mix Component generates one type of file :

- the **transaction-mix file**.

All together, the CAD system generates twelve transaction mix-files, four files per database. Each of the four files per database represent 24 transactions for a given record class.

The Evaluator's Report Component is comprised of two types of files:

- the **standard-text files**
- the **empirical-data files**.

The standard-text files present a narrative for the evaluator, providing instructions on how to interface the CAD generated test-database and test-transaction-mix files with MDBS. The text files also present a discussion for interpeting and analyzing the empirical data calculated by the CAD system. The empirical data files are a collection of tables that reflect information about the generated test-databases and generated test-transaction mixes based upon the user's input. A more detailed explanation of each of the files is presented in the discussion of its associated component.

3. The High-Level Program Structure of the CAD System

The CAD system has been designed utilizing a top-down strategy. The system is entitled as the **CAD Benchmark System**. At the highest level, 5

subordinate tasks are performed:

- Initialization.
- Receiving User Input.
- Initial Calculations,
- Creating Files, and
- Cleanup.

The first 3 tasks initialize the state of the CAD system. Within the 4th task, the major components of the CAD system are created. The 5th task performs a cleanup of all temporary files and allocated memory created by the CAD system once the CAD system has finished.

The high-level program structure can be described in three phases. The first phase is the **preparation** phase which encompasses the initialization of data structures, the solicitation of input data from the user, and the initial calculations. The second phase is the **files creation** phase which encompasses the generation of all the requisite test-database files, the test-transaction-mix files, and the report files. The third phase is the **cleanup** phase which encompasses system commands to purge all files no longer needed once the CAD system has completed its objective.

a. The Preparation Phase

The Preparation Phase begins the processing of the CAD system. Three tasks must be preformed prior to creating any of the three major components. Each task is discussed separately.

(1) Initialization. The initialization encompasses definition assignments, array initialization, and the declaration of data types. Variables are both global and local. Only the more important features of the initialization are addressed. The discussion covers the definitions, the arrays, key variables, and a structure representing the base table.

The more important definitions are those that reflect certain known factors about the methodology being implemented. These definitions are global and are shown in table 8. The maximum number of backends has been chosen to be 10. This is not a major constraint on the CAD system. Fixing the maximum number of backends has provided for fewer algorithms to be designed, coded and tested.

The more important arrays are those that hold information that is common to each of the major components. Two sets of arrays hold such information. One set of arrays holds the key attribute names and the other set of arrays holds the initial calculations. These arrays are also global and are shown in table 9.

One other array initialization important to the initial calculations is the least-common-multiple array known as the **lcm_table**. The lcm_table stores the least-common-multiple values for the integers 1 to 10. Recall that the system calculates the least common multiple for the number of backends in the system and the maximum number of backends presently allowed is 10.

TABLE 8. IMPORTANT GLOBAL DEFINITIONS

DESCRIPTION	NAME	ASSIGNMENT
Number of Cluster categories	num_clus_cat	9
Number of database sizes	num_db_sizes	3
Number of record sizes	num_rcd_sizes	4
Number of record classes	num_rcd_classes	4
Number of record factors	num_rcd_factors	4
Number of database factors	num_db_factors	3
Maximum number backends	max_num_be	10

TABLE 9. IMPORTANT GLOBAL ARRAYS

DESCRIPTION	NAME	VALUES
Template names	t_name[]	Templg Tempmedlg Tempmed Tempsmall
Descriptor names	INT_1_name[]	INTONELG INTONEMEDLG INTONEMED INTONESMALL
	INT_2_name[]	INTTWOLG INTTWOMEDLG INTTWOMED INTTWOSMALL
Record factor	rcd_factor[]	1 2 5 10
Database factor	dbase_factor[]	1 2 4
least common multiple	lc_table[]	1 2 6 12 60 60 420 840 2520 2520
four record sizes	four_rcd_sz[]	To Be Calculated
database sizes	dbase_sz[]	" "
Number of attributes per record size	num_attr_per_rcd_sz[]	" "

The base table depicted in Table 7 has been implemented utilizing a structure construct. Because only specific columns of the base table need to be accessed from time to time, the structure construct is the easiest to implement. Each of the first six columns is referenced as a unique independent array within the structure. The seventh column, the number of blocks per backend, is dependent on the number of backends in the system. The seventh column is described as a structure within a structure because of the nature of the response-time-reduction configurations. For each of the response-time-reduction configurations, the number of backends matches the configuration number up to the maximum number of backends described in the system. For each configuration, column 7 is described as a separate structure with each column entry for a specific backend being also a unique structure.

(2) Receive User Input. The Receive User Input module solicits the user for the three essential elements of information. The variable names for this information are:

DESCRIPTION	NAME	TYPE
Number of Backends	num_be	integer
Disk Track Size	dsk_trk_sz	integer
Maximum Disk Storage	max_dsk_storage	integer

(3) Perform Initial Calculations. Eight calculations are performed within the Initial Calculations task. Five of the calculations are functions that return values assigned to global variables. The remaining three calculations are procedures that determine values for the three global arrays. Each of the calculations is followed by an explanation.

The first calculation concerns the number of configurations. The number of configurations is equal to twice the number of backends less one.

THE NUMBER OF CONFIGURATIONS

Function		
Variable	Function name	Parameters Passed
num_config	Calc_config	num_be
num_config = 2 x num_be - 1		

The second calculation concerns the least common multiple. By accessing the lcm_table array in Table 9, the value of num_be being returned is the least common multiple. e.g., if num_be = 3, the value returned from the array is 6.

THE LEAST_COMMON MULTIPLE

Function		
Variable	Function name	Parameters Passed
lcm	Calc_lcm	num_be

The third calculation concerns the four record sizes. The first of the four records sizes is 1/2 the disk track size. The remaining three sizes are multiples of the first. Each remaining record size is calculated by dividing the first record size by the respective record factor (the rcd_factor array in Table 9).

THE FOUR RECORD SIZES

Procedure		
array	Procedure name	Parameters Passed
four_rcd_sz[]	Four_record_sizes()	dsk_trk_sz

The fourth calculation concerns the available disk storage. The available disk storage is 80% of the maximum disk storage. 20% of the disk storage is reserved for the MBDS directory.

THE AVAILABLE DISK STORAGE

Function		
Variable	Function name	Parameters Passed
avail_dsk_storage	Calc_avail_dsk_storage	max_dsk_storage

The fifth calculation concerns the database multiple. The database multiple is calculated by multiplying the least common mutiple by the number 32 by the largest record size (from four_rcd_sz array of Table 9). Chapter two presented the calculation for the database multiple in detail.

THE DATABASE MULTIPLE

Function		
Variable	Function name	Parameters Passed
dbm	Calc_database_multiple	lcm
dbm = lcm x 32 x four_rcd_sz[lg_rcd_sz].		

The sixth calculation concerns the database sizes. Recall from chapter II the total number of times the database multiple can be equally divided into the available disk storage results in a variable called **folds**.

THE DATABASE SIZES

Procedure		
Array	Procedure name	Parameters Passed
db_size[]	Database_sizes()	dbm, avail_dsk_storage

Thus, the largest database size equals the database multiple multiplied by the number of folds:

db_size[largest] = dbm x folds.

The remaining two databases (medium and small) are calculated by dividing the largest database size by the respective database factor.

The seventh calculation concerns the number of attributes per record size. The number of attributes per record size is calculated by dividing the record size by the numeric common divisor 10. The reason for this calculation is explained in the Generated Test-Database Component section.

THE NUMBER OF ATTRIBUTES PER RECORD SIZE

Procedure		
Array	Procedure name	Parameters Passed
num_attr_per_rcd_sz[]	Calc_num_attr_for_ea_rcd_sz	-

The eighth calculation concerns the key number of records. The key_num_rcds is a variable that identifies the number of records in the largest record class of the smallest database. This number of records is considered to be the key because every other record class size in each of the databases is a multiple of this number. The key_num_rcds variable is the determining factor for **all** calculations made with respect to the base record and block distribution table. Once the initial calculations are made the Create Files Phase begins.

THE KEY NUMBER OF RECORDS

Function		
Variable	Function name	Parameters Passed
key_num_rcds	Calc_key_num_records	-
key_num_rcds = (dbase_sz[small] / num_rcd_classes) / four_rcd_sz[large].		

b. The Create-Files Phase

The five modules of the Create Files Phase comprise the creation of the three major components of the CAD system: the generation of the test-database files. the generation of the test-transaction-mix files. and the generation of the report files. Most of the report files are generated simultaneously with the other two files. There is however one module dedicated solely to the generation of

one specific report. This section presents the high-level modules invoked for each of the major files.

(1) Generating the Test-Database Files. Upon completion of the initial calculations, the CAD system begins the generation of the test-database files. Three high-level modules are invoked:

- Make_template_file,
- Make_descriptor_files,
- Make_record_files.

Each of these modules in turn calls subordinate modules which generate the the appropriate requisite files. The details of these modules are presented in the Test-Database-Generation Component.

(2) Generating the Transaction-Mix Files. Upon completion of the test-database files, the CAD system begins the generation of the test-transaction-mix files. Just one high-level module is invoked for this task:

- Generate_trans_mixes.

The details of this module are presented in the Test-Transaction Mix Generation Component.

(3) Generating the Report Files. The CAD system invokes just one module to generate a specific report which is placed into 3 files. The module being invoked is:

- Format_test_benchmark_databases.

The specific report, which has its own dedicated module, provides the user with three tables of information. Each table is a file and is indicative of a database. The information in the table depicts the test configurations for a specific database size. Table 10 depicts the test configurations for a small database. The system parameters resulting in a specific configuration, for example, are:

available disk storage:	300	Megabytes
disk track size:	4000	bytes
number of backends:	3	

All of the other report files are comprised of information depicting data used in calculations, in relationships, and for attribute values. These report files are generated concurrently as the data is being generated within either of the other two major components. These report files include:

- the records-per-block-relationship table,
- the number-of-records-per-cluster-category table,
- the record-&-block-distribution table for each database,
- 4 transaction-mix files per database, and
- 4 transaction-mix workload files per database,

The standard text files are not generated by the CAD system. These files are actually part of the CAD system and are interleaved with the generated report files when the evaluator's report manual is assembled.

c. The Cleanup Phase

The cleanup phase is comprised of a number of system commands to purge all of the generated CAD files once the user has no further need for them.

TABLE 10. SMALL_DB_TEST CONFIGURATIONS

Configuration Number	Number of Backends	Record Size in Bytes	Number of Records per Backend	Mybytes per Backend	Database Size in Mbytes
1	1	2000	9372	18744000	
		1000	18744	18744000	
		400	46860	18744000	
		200	93720	18744000	74.976
2	2	2000	4686	9372000	
		1000	9372	9372000	
		400	23430	9372000	
		200	46860	9372000	74.976
3	3	2000	3124	6248000	
		1000	6248	6248000	
		400	15620	6248000	
		200	31240	6248000	74.976
4	2	2000	9372	18744000	
		1000	18744	18744000	
		400	46860	18744000	
		200	93720	18744000	149.952
5	3	2000	9372	18744000	
		1000	18744	18744000	
		400	46860	18744000	
		200	93720	18744000	224.928

B. THE TEST-DATABASE-GENERATION COMPONENT

The objective of the database component is to pass to a TEST directory all of the necessary files that represent the three database sizes, small, medium and large. Recall that the CAD constructs a tree directory for indexing the report and for each of the respective databases and its associated database and transaction-mix files. With respect to the generated test-database, the directory contains the three types of database files described by the attribute-based data model and utilized by MBDS. The three types of files are the **template** file, the **descriptor** file, and the **record** file. These files, generated by the CAD system, are manually loaded to MBDS.

The database component is described in two parts. The first part describes the generated test-database files and the second part addresses the database program modules.

1. The Generated Database Files

The CAD system generates three database sets, a large database, a medium database, and a small database. Each database may be described in terms of the generated files that represent it, i.e., the template file, the descriptor file, and the record files.

a. The Template File

Each of the database sets (small, medium, and large) shares a common single file known as the template file. This file contains four independent templates, with each template associated with a different record class. Each template contains the names of both the directory and non-directory attributes, and reflects the structure and the type of each of the record classes. By class we recall that Strawser [Ref. 5] creates four record classes, large, medium-large, medium and small. By type we mean the type of the attribute values, i.e., either string (s), integer (i), or floating number (f).

b. The Descriptor File

Each of the three database sets has its own unique descriptor file. The descriptor file contains indexing information for each directory attribute in the database. This indexing information takes two distinct forms. First, a directory attribute may be described by indexes which represent a list of the

possible values that the directory attribute may be assigned (a type-B attribute). Second. the directory attribute may be described by indexes which represent attribute-value ranges (a type-A attribute). Each descriptor file identifies the directory attributes: the template attribute. and the descriptor attributes. The template attribute receives as a value one of four template names describing the record size class: template large **(Templg)**, template medium-large **(Tempmedlg)**, template medium **(Tempmed)**, or template small **(Tempsmall)**.

The names given to the descriptor attributes identify the descriptor attribute itself, and the record class associated with it. Thus the names for the descriptor attribute is composed of three parts. The first part is entitled **INT** meaning integer. The second part is entitled either **ONE** or **TWO**. ONE associates the descriptor attribute to the nine cluster categories. TWO associates the descriptor attribute to the number of records per cluster. The third part associates the descriptor attribute to the record class. LARGE **(LG)**. MEDIUM LARGE **(MEDLG)**, . MEDIUM **(MED)**. OR SMALL **(SMALL)**. All together. there are eight possible attribute names:

INTONELG	INTTWOLG
INTONEMEDLG	INTTWOMEDLG
INTONEMED	INTTWOMED
INTONESMALL	INTTWOSMALL

Nine sets of values are assigned to the INTONExxx descriptor attributes, each set representing the range values associated with a specific cluster category. The set of values assigned to the INTWOxxx descriptor attributes are determined by the maximum range value found in the corresponding INTONExxx attribute, and the number of records per cluster for that specific cluster category. For example, given a database that has as part of its record configuration 9372 2000-byte records, table 11 reflects the range values for each of the descriptor attributes.

c. The Record File

Two distinct classes of record files are generated by the CAD system. The first class of record files represents the response-time-reduction (**rtr**) configurations. Each database (small, medium, and large) has its own unique rtr record file. The rtr file is associated with **all** of the rtr configurations of a given database because the size of that database does not change.

The second class of record files represents the response-time-invariance (**rti**) configurations. Each database has its own **set** of response-time-invariance record files. Each rti record file is associated with a specific rti configuration. Recall from chapter II that the number of rti configurations is dependent on the number of backends. We should note however that the number of rti record files per database set is the same.

The record file contains the information the user desires to have stored in the database. The information the CAD system places in each record takes five distinct forms. Each form is associated with a specific type of attribute.

TABLE 11

INTONELG ATTRIBUTE VALUE RANGES.
INTTWOLG ATTRIBUTE VALUE RANGES.

INTONELG Range of Values	INTTWOLG Range of Values
[1;344]	[1;4] [5;8] ... [341;344]
[345;866]	[345;350] [351;356] ... [861;866]
[867;1,562]	[867;874] [875;882] ... [1,555;1,562]
[1,563;2,432]	[1,563;1,572] [1,573;1,582] ... [2,423;2,432]
[2,433;3,476]	[2,433;2,444] [2,445;2,456] ... [3,465;3,476]
[3,477;4,694]	[3,477;3,490] [3,491;3,504] ... [4,681;4,694]

TABLE 11 CONT'D

INTONELG ATTRIBUTE VALUE RANGES.
INTTWOLG ATTRIBUTE VALUE RANGES.

INTONELG Range of Values	INTTWOLG Range of Values
[4,695;6,086]	[4,695;4,710] [4,711;4,726] ... [6,071;6,086]
[6,087;7,652]	[6,087;6,104] [6,105;6,122] ... [7,635;7,652]
[7,653:9,372]	[7,653;7,672] [7,673:7,692] ... [9,353:9,372]

Recall that the attribute-based data model considers data in terms of several constructs, two of which are directory keywords, and non-directory keywords. The first three forms of information are the values of the template attribute and the two descriptor attributes. These attributes are directory keywords. The other attributes of the record are non-directory keywords.

The first of the non-directory keyword attributes and the fourth form of information is described by an attribute entitled **Multiple**. The values of the multiple attribute are character strings representing how many times the database size has been multiplied, e.g., "One", "Two", "Three", . . ., etc. For example, if the database size has been doubled, the value of the multiple attribute is "Two". The effect of the multiple attribute is to double, triple, etc., the size of all existing clusters **without** defining any new clusters. Further detailed discussion of the

clustering concept is found in [Ref. 2]. In the response-time reduction record file, the multiple attribute **always** has a value of "One". The response-time-invariance record file multiple attribute's value ranges from "Two" to the character string representing the number of times the database size increases, e.g., for a system with 2 backends, the multiple attribute is assigned the values "One" and "Two". The database is doubled by duplicating the original record file and appending the duplicated file to the original file thus creating a new record file twice the size of the original. However, in the duplicated file the multiple attribute's value is changed from "One" to "Two", thus every record in the new database is unique. Table 2 in chapter II characterizes this assignment.

The fifth form of information is found in the attribute entitled **String**. The purpose of the string attribute is to represent sufficiently large nonprocessing data in order to make up the size of the record classes. All subsequent attributes are string attributes as well. The number of string attributes is dependent on the record sizes. Recall from Strawser's scheme [Ref. 2], four record classes are defined as large, medium-large, medium, and small. While the large record size is a function of the disk track size of the system, the remaining three sizes are multiples of the largest record size. The ratios are 1, 1/2, 1/5, and 1/10.

The number of string attributes is dependent upon the size of the respective record class. Because MBDS requires all attributes in a record to be the same size, a common divisor to all four record classes is selected. The CAD system

meets this requirement by soliciting the disk track size in increments of 1000. The increments of 1000 can best represent actual disk track sizes which are normally in increments of kilo bytes, where a kilo is actually 1024 bytes. In receiving the disk track size in increments of 1000, a common divisor of 10 ensures the number of attributes for each record size conforms to the record class ratios. For example, a system configured with a 4000 byte disk track size has 4 record sizes of 2000 bytes, 1000 bytes, 400 bytes, and 200 bytes. By selecting a common divisor of 10, we set the attribute size to 10-bytes per attribute. Table 12 show the number of 10-byte attributes corresponding to each record class.

TABLE 12. NUMBER OF 10-BYTE ATTRIBUTES PER RECORD CLASS.

Record Size in Bytes	Number of Attributes
2000	200
1000	100
400	40
200	20

A 2000 byte record has 200 attributes of which 196 are string attributes; a 200 byte record may have 20 attributes of which 16 are string attributes. Each string attribute's value is a string of x's, i.e., "Xxxxxxxxx". When the database is queried by the generated transaction mixes, the strings' values take on greater significance. In the transaction process the values are changed to more meaningful textual information. This process is explained later. Tables 13 and 14 depict the

TABLE 13

RECORD LAYOUT FOOTNOTES
^ The Attribute Template's value is a character string representing one of four record sizes, large, medium-large, medium, and small. In this example the record sizes represented are 2000, 1000, 400, and 200 byte records.
~ The names INTONExx and INTTWOxx represent the descriptor attributes.
~ The Attribute Multiple reflects the size of the database, i.e., One for size N, Two for 2N, Three for 3N, etc. A one-to-one mapping exists between the descriptor-Id set and the value of this attribute. The Multiple Attribute is used to double, triple, etc., the size of all existing clusters, without defining any new clusters.
* The number of string attributes is dependent upon the size of the respective record class, i.e., the 2000-byte record may have 200 attributes of which 196 are string attributes; the 200-byte record may have 20 attributes of which 16 are string attributes.

response-time reduction record layout for a small database for a system featuring 300 Megabytes of available disk storage with a 4000 byte disk track size.

d. The Generated Test-Database Report Files

Within the test-database-generation component two report files are created concurrently as two relationships are calculated. The first involves the records-per-block relationship. Recall from chapters I and II that the disk track size is the block size by which MBDS processes information from secondary memory on the backend to primary memory on the controller. Given the requirement to pass four different record sizes via the block, a records-per-block relationship is recognized. Table 15 depicts the relationship for a disk track size of 4000 bytes.

The second relationship involves record-per-cluster-category. This relationship is much like the first but a bit more complex. Recall from the section

TABLE 14

LOGICAL LAYOUT OF A RESPONSE TIME REDUCTION RECORD FILE

(small database)
(4000 byte disk track size)
(300 Mbytes of available disk storage)

Directory Keywords			Non-Directory Keywords			
Template[&]	INTONExx[~]	INTTWOxx[~]	Multiple[+]	String001 *	...	String--- *
TEMPLG	1	1	One	Xxxxxxxxx	...	Xxxxxxxxx
TEMPLG	2	2	One	Xxxxxxxxx	...	Xxxxxxxxx
...						
TEMPLG	9372	9372	One	Xxxxxxxxx	...	Xxxxxxxxx
TEMPMEDLG	1	1	One	Xxxxxxxxx	...	Xxxxxxxxx
...						
TEMPMEDLG	18744	18744	One	Xxxxxxxxx	...	Xxxxxxxxx
...						
TEMPMED	1	1	One	Xxxxxxxxx	...	Xxxxxxxxx
...						
TEMPMED	46,860	46,860	One	Xxxxxxxxx	...	Xxxxxxxxx
TEMPSMALL	1	1	One	Xxxxxxxxx	...	Xxxxxxxxx
...						
TEMPSMALL	93,720	93,720	One	Xxxxxxxxx	...	Xxxxxxxxx

TABLE 15. THE RECORDS-PER-BLOCK RELATIONSHIP

Record Sizes in Bytes	Records per Block
2000	2
1000	4
400	10
200	20

describing the CAD characteristics the need for nine cluster categories. This relationship describes the number of records per cluster category recognizing that each cluster category has a different number of blocks. Table 16 depicts this relationship.

TABLE 16. THE NUMBER OF RECORDS PER CLUSTER CATEGORY TABLE

Blocks per Cluster:	Record Size in Bytes:			
	2000	1000	400	200
2	4	8	20	40
3	6	12	30	60
4	8	16	40	80
5	10	20	50	100
6	12	24	60	120
7	14	28	70	140
8	16	32	80	160
9	18	36	90	180
10	20	40	100	200

2. The Generated Test-Database Program Modules

This section discusses the CAD program structure for the generation of the test databases. Recall from the section on Generating the Test-Database Files. that three high level modules are invoked to generate the test-database files:

- Make_template_file.
- Make_descriptor_files. and

- Make_template_file.
- Make_descriptor_files. and
- Make_record_files.

Each module is tasked to generate one of the three types of files germane to the test-database component.

a. The Generate-Template-File Module

The generate-template-file module creates one file containing four templates, one template for each record class. For each template the module writes to the file all the attributes' names: the template name. the descriptor names. the name multiple for the multiple attribute. and the strings' names. To determine the appropriate number of string attributes, the module accesses the number-of-attributes-per-record-size array (see Table 12).

b. The Generate-Descriptor-Files Module

The generate-descriptor-files module invokes five subordinate modules. Each of these modules is a procedure that is a prerequisite to the following modules with the exception of the Backend_table module. The five subordinate modules are:

- the Create_records_per_block_relationship_table module.
- the Create the record_per_cluster_category_table module.
- the Create the base record_block_distribution_table module.
- the Backend_table module. and
- the Write_descriptor_files module.

We discuss each of these modules in turn.

(1) The Records-Per-Block-Relationship Module. This module accesses the disk track size variable and the four record sizes array to calculate the relationships. A 2x2 array stores the relationship between the records per block and the record size themselves. For each record size, the number of records per block is calculated by dividing the disk track size by that record size.

(2) The Records-Per-Cluster-Category Module. This module accesses the records-per-block-relationship array and another array that stores the number of blocks per cluster category information to create a records-per-cluster-category array. For each cluster category, the number of bytes per record size is calculated by multiplying the records per block by the number of blocks per cluster.

(3) The Base Record-Block-Distribution Module. Table 7 depicts the logical layout of the base-record-&-block-distribution table (base table). To complete this table three data elements must be accessed: the records-per-cluster-category-table array, the key number of records variable, and the number of backends variable (num_be). Columns 1, 2, and 3 of the base table are a duplicate copy of the information stored in the largest record size of the records-per-cluster-category-table array.

Column 4, the total number of clusters per cluster category, involves a number of calculations. The first calculation involves summing the entries in column 3 of the base table. This value is the number of records per cluster category for all nine categories. The abbreviated variable name ttl_rcds_distr has been given to this variable for denoting the total number of

records for distribution. This value is then divided into the key number of records. If the division is without a remainder, then the total number of clusters (cluster size) for each cluster category is the same. Otherwise, the remainder is truncated and 1 cluster is added to the cluster size. When a remainder exists, a record overflow is incurred by the addition of the single cluster. The record overflow is calculated and then divided by the average number of records per cluster. This variable is named **clus_id**. This new value identifies the number of cluster categories that now must be decremented by a single cluster to insure the distribution of all records (key number of records). The cluster categories selected to be decremented are evenly distributed across all 9 categories.

Column 5, the total number of records per cluster category, is calculated by multiplying each row entry of column 2 by the corresponding row entry in column 3. Column 6, the total number of blocks per cluster category, is calculated by multiplying each row entry in column 2 by the corresponding row entry in column 3.

Column 7, the number of blocks per backend, is dynamic in nature and is dependent on the number of backends as well as directly associated with the configuration number. A subordinate module is invoked to calculate the appropriate values for a given case. For each entry in column 7, given the configuration, the value is equal to the total numbers of blocks per cluster category (column 6) divided by the number of backends in use.

(4) The Backend-Table Module. The backend-table module writes a report file that depicts the record-&-block-distribution table for each record class (large, medium-large, medium and small) within each database (small, medium, large). The base record-& -lock-distribution table is accessed and those columns which are dependent on the record factor, database factor, or both factors are multiplied by the appropriate factor with respect the record class and the database.

(5) The Write-Descriptor-Files Module. The write-descriptor-files module writes a descriptor file for each of the three databases. The module accesses the template names in the t_name array and writes them to the file. The module then accesses the total number of records per cluster category column (column 5) of the base table to determine the set of values for each of the 8 descriptor attributes. The module uses the mathmatical formula devised by Vincent [refV.p101] and the database and record factors presented as characteristics of the CAD to calculate the values.

c. The Generate-Record-Files Module

Each database has its own set of record files. The generate-record-files module creates a dynamic number of record files for each database dependent on the number of backends. Recall from the discussion of the record file that two distinct classes of record files are generated by the CAD system. One set is the rtr configuration records which comprises a single file. For each of the rti

configurations. a separate file is generated and whose size is a multiple of the rtr file. The generation of the record files is accomplished as in two steps.

In the first step, the initial file to be opened is the record file representing the response-time-reduction configurations. The size of this particular file is the base size for all the multiple size rti record files. For each of the record classes the appropriate number of records is written. First, the values for the descriptor attributes are written into the record and then the value "One" is written for the multiple attribute. For each of the number of string attributes (which is different for each record size) the value "Xxxxxxxxx" is stored. The total number of records to write is determined by summing all of the entries in the total-number-of-records-per-cluster-category column (column 5) and multiplying the sum by the record and database factors. Once this evolution has been completed for each record class, the file is closed.

In the second step, before a new file is opened, a copy of the last file written is made. Then a new file is opened and appended to the copy. For each of the response-time-invariance configurations, the multiple attribute value is increased by one. This evolution terminates when the requisite number of record files have been written. The entire process encompasses both the rtr file and the appropriate number of rti files. The two steps are performed for each database (large, medium, and small).

C. THE TEST-TRANSACTION-MIX-GENERATION COMPONENT

The objective of the transaction-mix component is to pass to the TEST directory all the transaction-mix files that represent the four record classes. Each of the three database sizes has four transaction-mix files, one for each record class. The transaction-mix component also generates a number of report files that portray information concerning the mixes themselves.

1. The Generated Test-Transaction-Mix Files

The CAD system generates three database sets, a large database, a medium database, and a small database. For each database three sets of files are generated. The first set of four files is the transaction-mix files, one for each record class. The second set (four files, one for each record class) is a printout of the transaction-mix report file. The third set (four files, one for each record class) is a printout of the test-transaction-mix-workload file. The second and third sets are used as an integral part of the evaluator's report. As each transaction request is generated for a specific record class within a specific database, the transaction-mix file, the transaction-mix report file, and the transaction-mix-workload file is written concurrently.

Appendix B includes the test-transaction-mix report and workload for the large record class of the small database for a system with 3 backends, a 4000 disk track size, and 375 Mbyte disk storage capacity. We encourage the reader to refer to the appendix as the remainder of the section is being read.

a. The Transaction-Mix Files

Vincent [Ref. 2:p. 104-120] has identified 30 transaction requests that encompass the five types of transactions found in MBDS, the **retrieval**, the **insert**, the **update**, the **delete**, and, the **retrieve-common**. The 30 transactions are further grouped into 7 request sets, each set designed to test the system with overhead-intensive or data-intensive type of operations. The 30 transactions are created for each record class within a database and comprise a separate file. All together, twelve transaction files are written.

The CAD system presently calculates only the first 24 transactions (6 request sets) of the 30 suggested by Vincent. These first 24 are the most important with regard to the system testing. The remaining 6 transactions (the 7th request set) are to be implemented at a later date.

b. The Generated Test-Transaction-Mix-Report Files

Two types of report files are generated within the test-transaction-mix component. The first type of report prints the actual transaction request sets with respect to each record class within each database. All together, twelve transaction mix files are generated.

The second type of report files generated in this component are the workload statistics for each of the requests in each of the request sets. Again (with respect to the record class and the database) twelve workload files are created.

2. The Generated-Test-Transaction-Mix-Program Modules

The Test-Transaction-Mix-Program Modules immediately follow the Test-Database Program Modules. Within the test-transaction mix component there is only one high-level module,

- Generate_trans_mixes.

Within this high-level module, the subordinate Generate-database-transaction-mixes module is invoked for each of the three databases, small, medium, and large.

a. The DB-Test-Transaction-Mix Module

For each of the four record classes this module opens three files (the transaction-mix file and the two report files) and invokes 6 subordinate modules. Each of the six subordinate modules generates one of the six request sets.

b. The Generate-Request-Set Modules

There are six generate-request-set modules. Each module represents one of the five types of transactions. Each transaction request is designed to test the system in terms of overhead or data intensive operations. For each request within each request set, the template and descriptor names are identified and the range values appropriate for that specific request are calculated. Instrumental to each calculation are the record and database factors.

In determining the values for the descriptor attributes, the CAD system targets specific cluster categories depending on the nature of the specific

request. For example, if the request involved 25% of the database, cluster category 4 would be the target. Within cluster category 4 the record range value for 25% of the database would be found.

Each request within a given request set is generated by a separate subordinate module. Within each of these subordinate modules, the test-transaction-mix report and the test-transaction-mix workload files are written. The same names and values calculated for the transaction mix are written to the test-transaction-mix-report file. For Request sets 1, 2, 3, 5, and 6 the workload of the request is described in terms of the number of clusters examined, the volume of the database accessed, and the volume of the database transacted. The workload for Request 4 is unique and will be described later.

(1) Generating Request Set 1. Request set 1 is comprised of three retrieval requests. Request #1 is overhead-intensive designed to retrieve a very small selection of records. Request #1 accesses the first cluster category and retrieves 1 cluster of records per each backend. Request #2 is data-intensive designed to access 2 cluster categories to retrieve a small selection of records. Request #3 is data-intensive designed to retrieve 25% of the database records while accessing just a fraction more than 25% of the database.

(2) Generating Request Set 2. Request set 2 is comprised of three update requests, all of which are data intensive. Request #4 identifies 1/8 of the database and updates STRING001's value from "Xxxxxxxxx" to "Oneeighth". Request #5 updates 1/4 of the database and updates STRINGS005's value from

"Xxxxxxxxx" to "Onequartr". Request =6 identifies 1/2 of the database and updates STRING010's value from "Xxxxxxxxx" to "Onehalf".

(3) Generating Request Set 3. Request set 3 is comprised of five retrieval requests, all of which are data intensive. Request #7 identifies the descriptor range value that is exactly 1/8 of the database and retrieves the records whose STRING001's value is "Oneeighth".

Requests #8, #9, and #10 retrieve portions of the database that are 1/8, 1/4, and 1/2 of the database, respectively. They identify the appropriate String attributes that were updated in Request Set 2. What they **do not** do is identify the descriptor range value that is exactly 1/8, 1/4, and 1/2 of the database, thus every cluster category (100%) of the database is examined! Request #11 identifies the descriptor range value for 1/2 of the database whose STRING010's value is "Onehalf".

(4) Generating Request Set 4. Request Set 4 is invoked just 3 times because it performs a retrieve-common operation of two record classes, i.e., of the large and medium-large, of the medium-large and medium, and of the medium and small. Request set 4 is comprised of three requests. Request #12 is overhead-intensive. Request #13 is data-intensive, and Request #13 is both overhead and data-intensive.

Request #12 uses the same values from Request #1 for its first retrieve. Its second retrieve value is a multiple of its first. Request #13 retrieves one half of the database from each record class. The STRING010 attribute's value

"Onehalf" is the determinant. Request #14 targets 1/8 of the database for both of its retrieve operations. Request #14 uses the same values from Request #7 for its first retrieve and makes the appropriate calculations for the second retrieve. Because no string attribute is targeted, descriptor attribute values are the determinants.

For each of the retrieve operations (the first retrieve being the source request and the second retrieve being the target request), the workload is described in terms of the number of clusters examined, the number of records accessed, and the number of relevant records to the retrieve-common request. For the retrieve-common request as an entire entity, the number of records retrieved resulting from the request is also depicted in the workload. All the workload data elements are written in table format as a part of the file.

(5) Generating Request Set 5. Request Set 5 is comprised of two insert requests. Request #15 inserts a record into an existing cluster. A record from the present record class is duplicated with the exception of the multiple attribute. The new value placed in the multiple attribute is one more than the maximum possible value with respect to the number of backends. E.g., if a system has had 3 backends whose last response-time invariance configuration has a multiple attribute value of "Three", the new value would be "Four".

Request #16 inserts a record into a **new** cluster. A record from the first cluster category is duplicated with the exception of the INTWOxxx descriptor attribute (which identifies the number of records per cluster). By

determining the maximum range value for this descriptor in this specific cluster category. a new value larger than the maximum range value is selected as the appropriate value. By defining a new descriptor whose value is not within the range limits of those clusters, a new cluster is created for that record to be inserted.

(6) Generating Request Set 6. Request Set 6 is comprised of eight delete requests. Request #17 is overhead-intensive, Request #18 is both data and overhead-intensive, and Requests #19 through #24 are data-intensive.

Request #17 and #18 mirror retrieve Request #1 and #2, respectively. Request #20 - #24 mirror retrieve Request #7 - #11, respectively. In all of these requests, the workload is exactly the same as the mirrored request. Request #19 corresponds to Request #3's workload. However the last 25% of the database is deleted. Again the target cluster category is identified (category 8) and the appropriate value is calculated.

D. THE REPORT COMPONENT

The evaluator's report component provides the user with a detailed explanation of the benchmarking process. The report is comprised of standard text files interleaved with a number of tables encapsulating data and statistical information about the system defined by the user's input. Given the three essential elements of information solicited from the user, the CAD system produces a report that provides information describing four main topics: the

generated test databases, the generated test-transaction mixes, the TEST directory and all the files contained therein, and the instructions on how to integrate the CAD generated test files with MBDS.

The reports describing the generated test databases are:

- the test-configurations report for each of the given databases,
- the record-block-relationship report for the base table,
- the records-per-cluster-category-relationship report for the base table, and
- the record-&-block-distribution-table report for each record class for each given database.

The reports describing the generated test-transaction mixes are:

- the transaction-mix report for each record class within a given database, and
- the transaction-mix workload for each record class within a given database.

Appendix B includes examples of all the generated report files.

The standard text files are presently being drafted. Once drafted and placed within the CAD system, calls are to be coded to interleave the text files with the generated report files. The purpose of the text files is two-fold. The first objective is to explain exactly the nature of the generated test report files found within the TEST directory. The second objective is to explain interfacing the generated test files with MBDS.

Most of the CAD report files are generated concurrently with the test files within the other two major components. However, the CAD system does call one high-level module dedicated to generating three specific sets of tables, each table

being a unique report file. Each table (or file) depicts the test configurations for a given database. small. medium. or large.

For each database size. the module formats the response-time-reduction configurations followed by the response-time-invariance configurations. Several data elements must be accessed: the four-record-sizes array (four_rcd_sz), the database-sizes array (dbase_sz), the number of backends (num_be). and the number of configurations (num_config). For the rtr configurations the database size (in Mbytes) is accessed from the database-sizes array. Incrementing through each configuration number with respect to the number of backends. the number of records per backend is calculated as well as the Mbytes per backend for each record class. The data is written to a file in table format.

For the rti configurations the same algorithm applies with the exception of increasing the database size (double, triple. etc.) and with respect to the configuration number.

IV. CONCLUSIONS

Database management systems have taken three approaches to information processing, the traditional mainframe-based approach, the software single-backend system, and the software multiple-backend system. The performance and upgrade problems identified with the first two approaches are overcome by the third approach, the software multiple-backend system, by providing gains through specialization of the database operations on dedicated, multiple backends.

Two goals of the software multi-backend database system are to overcome the performance problems and upgrade issues of the traditional mainframe-based and the conventional software single-backend database systems. The first goal is to produce a reciprocal decrease in the response times of the user transactions by increasing the number of backends while the size of the database and the size of the responses to the transactions remain constant. The second goal is to produce invariant response times for the user transactions by increasing the number of backends proportionally to the increase of the transaction responses. The first goal allows the multiplicity of the backends of the database system to be directly related to the performance gains of the database system in terms of the response-time reduction. The second goal enables the multiplicity of the backends of the

system to be directly related to the capacity growth of the system in terms of response-time invariance.

To verify the aforementioned performance and growth-capacity claims, Vincent [Ref. 2] has formulated a benchmarking methodology for software multiple-backend database systems. In this thesis, we have presented a computer-aided design (CAD) system for the generation of test databases and test-transaction mixes that can be used for the purpose of benchmarking parallel, multiple-backend computer systems, specifically the Multiple-Backend Database System (MBDS).

To fully understand the implementation of the CAD system, we have reviewed the essence of Vincent's methodology, specifically the test-databases and the test-transaction-mix design factors. We have also described the prototyped multi-backend database system at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, California, including a discussion of the attribute-based data model and the attribute-based data language.

The most salient features of the CAD system are two characteristics that are an integral part of each of the three major components. The first characteristic involves the creation of a number of factors that serve as multipliers for entities. Specifically there are two factors, one for the database and one for the record classes. The second characteristic involves the creation of a base-record-&-block-distribution table whose numeric entries reflect the upper bounds on the number of records, blocks, and clusters formed and distributed across the backends of

MBDS. The values in the table are used to generate the appropriate number of records needed for a precalculated database size. They are also used to determine the range values of descriptor files, and they serve to determine the values in the transaction mixes as well.

The CAD system has been designed to receive a minimal amount of information concerning a specific database system from the user an to transform this input into the requisite test databases and test-transacti mixes. The framework of the CAD system is built around three major compo nts, the test-database-generation component, the test-transaction-mix-generation component, and the evaluator-report component. The final output resulting from the combined work of the major components is the CAD system's generation of two sets of files placed in a TEST directory for the user. The first set of files is expressly for testing MBDS. The second set of files comprises a number of reports describing the test databases and the test-transaction mixes. In conjunction with the second set of files, the CAD system interleaves a number of standard text files that present a narrative for the evaluator providing instructions on how to interface the CAD generated test-database and test-transaction-mix files with MDBS. The text files also present a discussion for interpeting and analyzing the empirical data calculated by the CAD system. The text files are presently being drafted and are to be incorporated into the CAD upon completion.

The CAD system described in this thesis is a first version. Presently the versin generates the test databases and test-transaction-mix files and places them into a

TEST directory. The system provides the user with an evaluator's report that includes instructions on how to manually load MBDS with the generated test files. The second version is to be integrated with MBDS allowing the testing process to be controlled and managed by the CAD system. The third version will add components to collect statistics (e.g. response times) for the different tests and calculate statistics (i.e., mean and standard deviation of tests, response-time reductions and response-time invariances) that measure the performance of MBDS.

Task: CAD_Benchmark_System

```
/*The following are global arrays                                          */
/*    t_name[]: an array of the four template names, Templg, Tempmedlg,    */
/*              Tempmed, and Tempsmall                                      */
/*    INT_1_name:[] an array of the four descriptor ONE names. INTONELG,   */
/*              INTONEMEDLG, INTONEMED, and INTONESMALL                     */
/*    INT_2_name[]: an array of the four descriptor TWO names. INTTWOLG,   */
/*              INTTWOMEDLG, INTTWOMED, and INTTWOSMALL                     */
/*   Multiple[]: an array of 11 multiple names."One","Two"..."Eleven"      */
/*    rcd_factor[]: an array of the record factors, 1, 2, 5, and 10        */
/*    dbase_factor[]: an array of the database factors. 1, 2, and 4        */
/*    num_attr_per_rcd_sz[]: an array of the number of attributes per      */
/*              record size (class)                                         */
/*    rcd_per_blk_rel_tbl[r,c]: a 2x2 array of the record-block relation   */
/*    rcd_per_clus_cat_tbl[r,c]: a 9x4 array of the cluster category and   */
/*              the record-block relationship                               */
/*    base_rcd_&_block_distr_table                                         */
/*    lcm_table: an array of lcm's indexed by the number of backends       */


perform Receive_user_input(num_be, dsk_trk_sz, max_storage_dsk_sz);

perform Initial_calculations(num_be, dsk_trk_sz, max_storage_sz);

perform Make_Template_Files();

perform Make_Descriptor_Files(num_be, dsk_trk_sz,key_num_rcds,
                                        max_dsk_storage);

perform Make_Record_Files(num_be, key_num_rcds);

perform Generate_Trans_Mix_Files(num_be);

perform Format_test_bmark_dbs(num_be, num_config);

perform Clean_Up();

end task;
```

```
procedure Receive_user_input( output: num_be, dsk_trk_sz,
                              max_storage_dsk_sz);
        perform Get$Number_of_Backends (num_be);

        perform Get$Disk_Track_Size (dsk_trk_sz);

        perform Get$Disk_Storage_Size (max_storage_dsk_sz);

end procedure
```

```
module Get$

procedure Number_of_Backends ( output: num_be);

                /* num_be is the number of backends solicited    *
                /* from the user                                 */
end procedure;


procedure Disk_Track_Size ( output: dsk_trk_sz);

                /* dsk_trk_sz is the disk track size of the      */
                /* system solicited from the user                */
end procedure;


procedure Disk_Storage_Size ( output: max_storage_dsk_sz);

                /* max_storage_dsk_sz is the maximum size of     */
                /* the disk storage (in bytes). This value is    */
                /* inputted by the user                          */
end procedure;
```

```
procedure Initial_calculations()
        perform Calculate$Configurations (num_be, num_config);
        perform Calculate$lcm(num_be, lcm);
        perform Calculate$Four_Record_Sizes (dst_trk_sz);
        perform Calculate$Avail_Disk_Storage(max_storage_dsk_sz);
        perform Calculate$Database_Multiple(lcm);
        perform Calculate$Database_Sizes(dbm, avail_dsk_storage);
        perform Calculate$Calc_num_attr_for_ea_rcd_sz();
        perform Calculate$Num_attr_for_ea_rcd_sz();
        perform Calculate$Key_num_rcds();
end procedure
```

```
module Calculate$

function Configurations ( input: num_be,
                          output: num_config);

                num_config = 2*(num_be) - 1;

                /* num_config is the number of test configurations      */
                /* for each of the databases, small, medium, and        */
                /* large                                                */
end procedure;



function lcm ( input: num_be, lcm_info_ptr)
               output: lcm);

                    Access table and select the corresponding lcm value
                    for the respective num_be

                /* lcm is the least common multiple based on the        */
                /* number of backends, num_be                           */
end procedure;



procedure Four_Record_Sizes ( input: dsk_trk_sz, rcd_factor[],
                              output: four_rcd_szs [lrs, mlrs, mrs, srs])

                Determine the four record sizes based on Strawser's
                scheme

                four_rcd_sz[0] = dsk_trk_sz / 2;

                for every other four_rcd_sz[i], i ranging 1 to 3 do

                        four_rcd_sz[i] = four_rcd_sz[0] / rcd_factor[i];
                end for


                /* the four record sizes are stored in a global array   */
                /* four_rcd_sz[], lrs is large record size,             */
                /* mlrs is the medium-large record size,                */
                /* mrs is the medium record size                        */
                /* srs is the small record size                         */
                /* rcd_factor[] is a global array                       */
end procedure;
```

```
function Avail_Disk_Storage ( input: max_storage_dsk_sz,
                              output: avail_dsk_sz);

                avail_dsk_sz = 80% of max_storage_dsk_sz

                /* avail_dsk_sz is that portion of the maximum disk storage
                /* that is reserved for the database per backend
end procedure;



function Database_Multiple ( input: lcm,
                             output: dbm);

                lg_rcd_sz = four_rcd_sz [0 ;

                dbm = (lcm x 32 x lg_rcd_sz);

                /* dbm is the database multiple */
end procedure;



procedure Database_Sizes ( input: dbm, avail_dsk_sz,
                           output: dbase_sz[lg_db_sz,
                              med_db_sz, small_db_sz]);

/* The following variables are local */
        folds
        lg_db_sz

                folds =  avail_dsk_sz / dbm;      /* discard the remainder */
                dbase_sz[0] = (dbm) x folds;
                lg_db_sz =dbase_sz[0];
                dbase_sz[1] = 1/2 of lg_db_sz;
                dbase_sz[2] i= 1/4 of lg_db_sz;

                /* dbase_sz[] is a global array                              */
                /* lg_db_sz is the large database size,, db_sz[0]            */
                /* med_db_sz is the medium database size, db_sz[1]           */
                /* small_db_sz is the small database size. db_sz[2           */
end procedure;
```

```
procedure Calc_num_attr_for_ea_rcd_sz( output: num_attr_per_rcd_sz    ):

        Determine a common divisor to all four record sizes.
        Divide each of the record sizes by the divisor.
        The number 10 may be satisfactory.

        /* num_attr_per_rcd_sz is a data construct containing          */
        /* the respective number of attributes for each of the         */
        /*  record sizes: large, med-lg, medium, small                 */

end procedure;


function Calc_key_num_rcds ( output: key_num_rcds);

        key_num_rcds = (dbase_sz[small_db_sz] num_rcd_classes)
                                                    four_rcd_sz[lg_rcd_sz]);

                /* dbase_sz[] is a global array                        */
                /* num_rcd_classes is a global constant                */
                /* four_rcd_sz[] is a global array                     */
```

```
procedure Make_Discriptor_files( input: num_be, dsk_trk_sz,
                                 key_num_rcds, max_dsk_storage,
                                 output: three descriptor files):

        perform Initialize_clus_cat_table():

        perform Create_record_per_block_relationship_table(dsk_trk_sz);

        perform Create_rcd_per_cluster_cat_table():

        perform Create_base_record_block_distribution_table(num_be, key_num_rc

        perform Backend_table(num_be);

        perform Write_descriptor_files(num_be, key_num_rcds);

end procedure;
```

```
procedure Create_records_per_block_relationship_table ( input dsk_trk_sz.
                          output rcd_per_clus_cat_table r,c );

 * The following code creates the Records per Block Relationship Table *
 *  which is a global array                                            *

/* The following variables are local                                   */
/*      block: the disk track size for memory transfer                 */

        block = dsk_trk_sz
                for each record size, four_rc_sz[i] do
                        rcd_per_blk = block/rcd_sz[i]
                        write rcd_sz[i], rcd_per_blk
                end for;

end procedure;



procedure Create_rcd_per_cluster_catagory_table ( output:
                                                  rcd_per_clus_cat_table[r,c] );


        access block per cluster information (CCI array)
        and the record-block relationship array which are global

        scalar multiply the records_per_block_relationship_table x (CCI array)
        and store in rcd_per_clus_cat_tbl[] array

        write the r/cc table to a file;

end procedure;
```

```
procedure Create_base_record_&_block_dist_table ( input: num_be
                                                  key_num_rcds.
                                  output: base_rcd_&_blk_distr_table r.c ):

        /* base_rcd_&_blk_distr_table r,c  is a global structure       *

        /*     (col 0)Record_sz_in_byte_column_array                   *
        /*     (col 1)Number_of_blocks_per_cluster_column_array        *
        /*     (col 2)Number_of_records_per_cluster_column_array       *
        /*     (col 3)Total_number_of_clusters_column_array            *
        /*     (col 4)Total_number_of_records_column_array             *
        /*     (col 5)Total_number_of_blocks_column_array              *
        /*     (col 6)A structure to another (dynamic) table dependent */
        /*       on "m" configurations                                 */

        perform Complete_column_0(base_rcd_&_blk_distr_table[r,c]);

        perform Complete_columns_1_&_2(base_rcd_&_blk_distr_table[r,c]);

        perform Complete_column_3(base_rcd_&_blk_distr_table[r,c]);

        perform Complete_columns_4(base_rcd_&_blk_distr_table[r,c]);

        perform Complete_columns_5(base_rcd_&_blk_distr_table[r,c]);

        perform Complete_column_6(base_rcd_&_blk_distr_table[r,c]);

end procedure;
```

```
procedure Complete_columns_1_&_2()

        access rcd_blk_distri_table r,c

        copy rcc_table r,c  (large record size size only) to columns 2 & 3 in
        rcd_&_blk_distri_table r,c;

        /* writes r/cc table to base_rcd_&_blk_table */

end procedure;
```

```
procedure Complete_column_3( input: key_num_rcds,
                             output: base_rcd_&_blk_distr_table r.c ):

/* Completes column 3 of the base table                                  *
/* Performs the cluster distribution for the nine cluster catagories */

/* The following are local variables:                                    */

/*      ttl_rcds_distr is the total records per cluster summed over  */
/*         all nine cluster categories                               */
/*      case_id identifies which cluster categories are to be decre- */
/*         mented by 1 cluster to ensure proper distribution         */
/*      equal_dist is a flag indicating whether or not all 9 cluster */
/*         categories are to receive the exact same number of        */
/*         clusters                                                  */
/*      cluster_size is the number of clusters per cluster category  */

        ttl_rcds_distr =  Calculate_trd();

        cluster_size =  Calculate_cluster_size(key_num_rcds.ttl_rcds_dist,
                              case_id, equal_dist);

        access the column 3 of the rcd_&_blk_distri_table ,
        "total number of clusters" column

        base_rcd_&_blk_distr_table[r,3]

        for each cluster catagory in the above column, r ranging from 1 to 9 do
          base_rcd_&_blk_distr_table[r,3  = cluster_size:
```

```
    if equal_dist is false

            access the num_of_rcds_per_cluster column (column 2)
            in the rcd_&_blk_distri_table r,2

            for case_id(i)

             i = 1 subtract 1 from cluster catagory 5, 1 to 9 catagories

             i = 2 subtract 1 from cluster catagories 1 and 9

             i = 3 subtract 1 from cluster catagories 1,5 and 9

             i = 4 subtract 1 from cluster catagories 1,2,8 and 9

             i = 5 subtract 1 from cluster catagories 1,2,5,8 and 9

             i = 6 subtract 1 from cluster catagories 1,2,3,7,8 and 9

             i = 7 subtract 1 from cluster catagories 1,2,3,5,7,8 and 9

             i = 8 subtract 1 from cluster catagories 1,2,3,4,6,7,8 and 9

            end case
end procedure
```

```
function Calculate_trrd ()

        sum the total number of records in the large rcord size column
        of the rcd_per_clus_cat_table(r/cc table); this value is the
        ttl_rcds_dist (total records for distribution)

        /* use for loop, index from 1 to 9 and sum row entries assigning */
        /* sum total to ttl_rcds_dist                                    */

end procedure;

function Calculate_cluster_size ( input: key_num_rcds, ttl_rcds_dist,
                                     case_id, equal_dist,
                                  output: cluster_size,
                                     equal_distribution, case_id);

/* The following are local variables                                    */

/*    rcd_overflow is the result of unequal distribution               */
/*    record_deficit is the number of  records short for               */
/*       even cluster distribution                                     */
/*    avg_num_rcds_per_cluster                                         */

        initialize equal_distribution to false;
        num_clus_cat = 9;

        cluster_size = (key_num_rcds / ttl__rcds_dist);
        rcd_overflow = key_num_rcds - (cluster_size x ttl_rcds_dist);

        if (rcd_overflow == 0)

                equal_distribution = true;

        else
                cluster_size = cluster_size - 1;
                equal_distribution = false;
                avg_num_rcds_per_cluster = ttl_rcds_dist / num_clus_cat;
                record_deficit = ttl_rcds_dist - rcd_overflow;
                case_id = record_deficit / avg_num_rcds_per_cluster;

        /* rcd_overflow is the record overflow if the number of          */
        /*    records cannot be evenly distributed into an equal number of */
        /*    clusters considering the original cluster_size calculation  */

        /*avg_num_rcds_per_cluster is the average number of records per cluster
        /*num_clus_cat is the number of cluster catagories (global constant)
        /*case_id is the case identifier which determines which cluster catagorie
        /*  will be adjusted in size, i.e. decremented by a single cluster

end procedure;
```

```
procedure Complete_col_4 ()

        access rcd_&_blk_distri_table;

        to calculate the value for each row entry in column 4,
        multiply the corresponding row entry from column 2 by
        the corresponding row entry from column 3;

end procedure;


procedure Complete_col_5()

        access rcd_&_blk_distri_table;

        to calculate the value for each row entry in column 5,
        multiply the corresponding row entry from column 1 by
        the corresponding row entry from column 3;

end procedure;
```

```
procedure Complete_col_6(num_be)

 * The following variables are local                              *

/*  num_rtr_configs is the number of rtr configurations    */
/*  ttl_num_blks is the row entry, total number of blocks  */
/*  extra_blocks is the number of blocks remaining follow- */
/*     ing even distribution of total number of blocks     */
/*     across the backends                                 */
/*  block_distribution is the block distribution per       */
/*     backend                                             */

       access rcd_&_blk_distri_table(r,6);

       num_rtr_configs = num_be;

       do the 1st configuration

              copy col_5 to col_6

       if m > 1 do

        for configurations(i) ranging from 2 to num_rtr_configs do

              create i columns

              for each row entry,r do r ranging from 1 to 9 -
                   /* 9 times, once for each cluster catagory */

                   read ttl_num_of_blks (row entry) of rcd_&_blk_distri_table

                   extra_blocks = ttl_num_of_blks MOD i;

                   block_distribution =  ttl_num_blks / i;

                   for ea be_column entry,c do c ranging from 1 to i

                         be_col_entry(r,c) = block_distribution;

                   end for
```

```
                    if extra_blocks = 0
                            /* do nothing */
                    else
                            for x = 1 to extra_blocks
                                be_column [r,y] = be_column [r,y] - 1
                                y = y - 1;
                                if y = i
                                   y = 1;
                                x = x + 1;
                            end for
                     end if
                end for
        end for
end procedure
```

```
procedure Write_discriptor_files( output: 3 descriptor files):

 * t_name is a global array containing the four template names                *
 * INT_1_name is a global array containing the four INTxx1 discriptor names   *
 * INT_2_name is a global array containing the four INTxx2 discriptor names   *
 * rcd_factor is a global array containing the four record factors            *
 * dbase_factor is a global array containing the three database factors       *

      for each dbase_factor [db], db ranging from 1 to 3 do

            write to a file
              database_id_name = TEST
              TEMP c s
              for t_name[j] j ranging from 1 to 4
                    write the character !, and t_name[j]
              end for
              the character @


            access base_rcd_&_blk_distr_table[r,c]

            for INT_1_name[i], i ranging from 1 to 4 do

                  write the INT_1_name[i], a, i

                  /* "a" is the type of attribute, i stands for integer */

                  first_value = 1;

                  second_value = 0;

                   factor = rcd_factor[i * dbase_factor[db]

                  for base_rcd_&_blk_table r,4 , r ranging from 1 to 9 do

                    temp_value = base_rcd_&_blk_distr_table[r,4] * factor

                    second_value = second_value + temp_value

                    write first_value, second_value

                    first_value = second_value + 1

                  end for

                  write the character @

            end for
```

```
            for INT_2_name i   i ranging from 1 to 4 do
               write the INT_2_name[i], a, i
               /* "a" is the type of attribute, i stands for integer *
                factor = rcd_factor[i] * dbase_factor[db]
               w = 0
               for each entry in base_rcd_&_blk_distri_table [r,2] r ranging
               from 1 to 9 do
                    x = base_rcd_&_blk_distri_table[r,2] * rcd_factor[i]
                    tnc = base_rcd_&_blk_distri_table[r,3] * db_factor[db]
                    for each value y, y ranging from 1 to tnc do
                            first_value = w + (x*y) - (x - 1)
                            second_value = w + (x*y)
                            write first_value, second_value
                    end for
                    w = w + x*y
               end for
               write the character @
         end for
         write the character $
         send file to directory placing routine
      end for (dbase_factor loop)
end procedure
```

```
procedure Write_record_files( input: num_be.key_num_rcds.
                              output: record files):

 * The following are global data elements:                              *
 * num_attr_per_rcd_sz is a global array containing                     */
       the number of attributes per record class                       */
/* key_num_rcds is a  global variable; the number of records from       */
/*       the large record class from the small database set.            */
/* num_be is the number of backends                                     */
/* t_name is a global array of the template names                       */

/* The following are local data elements:                               */
/* max_string_value is the max number of string attributes for a given  */
/*       record size                                                    */
/* str_num is a loop index meaning number of string attributes          */

        for each dbase_factor [db], db ranging from 1 to 3 do

              for each multiple[i], i ranging from 1 to num_be do

                write to a file
                  database_id_name = TEST
                  the character @

                  for t_name[j] j ranging from 1 to 4  do

                        factor = rcd_factor[j] * db_factor[db]
                        for value = 1 to key_num_rcds * factor do

                                write t_name[j]
                                write value, value, multiple[i],

                                max_string_value = num_attr_per_rcd_sz[j] - 4
                                for str_num = 1 to max_string_value do

                                        write Xxxxxxxxx

                                end for

                        end for

                        write the character @

                end for

                if multiple[i] > 1  then
                        get previous file written: append current file
                        to previous file

                send file to directory routine

end procedure
```

```
perform Generate_trans_mixes( input: num_be.
                              output: 12 transaction mix files):

/*  Twelve transaction mix files are generated, one per record class  */
/*    for each of the three databases                                  */

/*  Twelve transaction mix report files are generated, one per        */
/*    record class for each of the three databases                     */

/*  Twelve transaction mix workload files are generated, one per      */
/*    record class for each of the three databases                     */

        For each of the three databases, db do

                Perform Gen_db_trans_mixes()

end procedure;
```

Procedure Gen_db_trans_mixes()

```
        for each of the four record classes,  rs do

                open a file for the transaction mixes;

                open a file for the transaction mixes report;

                open a file for the transaction mixes workload;

                factor = dbase_factor[db] * rcd_factor[rs];

                perform Gen_reqset1();
                perform Gen_reqset2();
                perform Gen_reqset3();
                perform Gen_reqset4();
                perform Gen_reqset5();
                perform Gen_reqset6();

                close all files

        end for loop

end procedure;
```

```
Procedure Gen_reqset1()

        print table headings for the report file and workload file

        perform Request1();
        perform Request2();
        perform Request3();

end procedure;
```

Request1()

```
        OBJECTIVE: Retrieve

        Assign the appropriate template and descriptor names.

        Given the number of backends determine the minimum number of
          records to be retrieved such that each backend provides the
          same number of records. The target cluster category is 1.
        The records to be retrieved should be taken from the middle
          of the cluster category.
        The values selected  are for the INTONExx descriptor.

        For the workload:
        Calculate the number of records accessed by counting all the
          records in the first category.
        Calculate the number of records retrieved by multiplying the
          number of backends by the number of records per cluster for
          cluster category 1. Remember only 1 cluster per backend is
          retrieved.

end procedure;
```

Request2()

```
        OBJECTIVE: Retrieve

        Request2 has been stubbed.

end procedure;
```

Request3()

```
        OBJECTIVE: Retrieve

        Assign the appropriate template and descriptor names.

        Determine 25% of the database, and calculate the INTTWOxx descriptor
          value.
```

For the workload:

Determine which cluster category earmarks the INTWOXX value and calculate the volume of the database accessed by counting all the records in that cluster plus all preceding clusters then divide by the total number of records in all 9 categories.

The volume of the database retrieved is 25%.

Procedure Gen_reqset2()

```
        print table headings for the report file and workload file

        perform Request4();
        perform Request5();
        perform Request6();

end procedure;
```

Request4()

```
        OBJECTIVE: Update

        Assign the appropriate template and descriptor names.

        Calculate 1/8 of the given database and assign that value to
         the INTWOxx descriptor attribute.

        For the workload:
        Determine 1/8 of the database.  Determine which cluster
         category earmarks 1/8 of the database and calculate
         the volume of the database accessed by counting all the records
         in that cluster plus all preceding clusters, then divide that
         sum by the total number of records in all categories.
        The volume of the database updated is 12.50%.

end procedure:
```

Request5()

```
        OBJECTIVE: Update

        Assign the appropriate template and descriptor names.

        Calculate 1/4 of the given database and assign that value to
         the INTWOxx descriptor attribute.

        For the workload:
        Determine 1/8 of the database.  Determine which cluster
         category earmarks 1/4 of the database and calculate
         the volume of the database accessed by counting all the records
         in that cluster plus all preceding clusters. then divide that
         sum by the total number of records in all categories.
        The volume of the database updated is 25.00%.

end procedure;
```

```
Request6()

        OBJECTIVE: Update

        Assign the appropriate template and descriptor names.

        Calculate 1/2 of the given database and assign that value to
         the INTWOxx descriptor attribute.

        For the workload:
        Determine 1/2 of the database.  Determine which cluster
         category earmarks 1/2 of the database and calculate
         the volume of the database accessed by counting all the records
         in that cluster plus all following clusters, then divide that
         sum by the total number of records in all categories.
        The volume of the database updated is 50.00%.

end procedure;
```

Procedure Gen_reqset3()

```
        print table headings for the report file and workload file

        perform Request7();
        perform Request8();
        perform Request9();
        perform Request10();
        perform Request11();

end procedure;
```

Request7()

```
        OBJECTIVE: Retrieve

        Half of the database is to be accessed and 1/8 is to be retrieved.

        Assign the appropriate template and descriptor names.

        Calculate 1/2 of the given database and assign that value to
         the INTONExx descriptor attribute.

        Assign  STRING001 attribute the value Oneeighth.

        For the workload:
        Determine 1/2 of the database.  Determine which cluster
         category earmarks 1/2 of the database and calculate
         the volume of the database accessed by counting all the records
         in that category plus all preceding categories. then divide that
         sum by the total number of records in all categories.
        The volume of the database retrieved is 12.50%.

end procedure;
```

Request8()

```
        OBJECTIVE: Retrieve

        All of the database is to be accessed and 1/8 is to be retrieved.

        Assign the appropriate template name.

        Assign  STRING001 attribute the value Oneeighth.

        For the workload:
        The volume of the database accessed is 100.00%.
        The volume of the database retrieved is 12.50%.

end procedure;
```

Request9()

OBJECTIVE: Retrieve

All of the database is to be accessed and 1/4 is to be retrieved.

Assign the appropriate template name.
Assign STRING005 attribute the value Onequartr.

For the workload:
The volume of the database accessed is 100.00%.
The volume of the database retrieved is 25.00%.

end procedure;

Request10()

OBJECTIVE: Retrieve

All of the database is to be accessed and 1/2 is to be retrieved.

Assign the appropriate template name.
Assign STRING010 attribute the value Onehalf.

For the workload:
The volume of the database accessed is 100.00%.
The volume of the database retrieved is 50.00%.

end procedure;

Request11()

OBJECTIVE: Retrieve

Half of the database is to be accessed and 1/2 is to be retrieved.

Assign the appropriate template and descriptor names.

Calculate 1/2 of the given database and assign that value plus 1 to the INTTWOxx descriptor attribute.

Assign STRING010 attribute the value Onehalf.

For the workload:
Determine 1/2 of the database. Determine which cluster category earmarks the INTWOXX value of the database and calculate the volume of the database accessed by counting all the records in that category plus all following categories, then divide that sum by the total number of records in all categories.
The volume of the database retrieved is 50.00%.

end procedure;

Procedure Gen_reqset4()

```
        print table headings for the report file and workload file

        perform Request12();
        perform Request13();
        perform Request14();

        Note that the workload for Request4 involves the following:
         Number of clusters examined by the first (source) retrieve
         Number of records accessed by the source retrieve
         Number of records relevant to the source retrieve
         Number of clusters examined by the second (target) retrieve
         Number of records accessed by the target retrieve
         Number of records relevant to the target retrieve
         Size of the resulting record set (in number of records)

end procedure;
```

Request12()

```
        OBJECTIVE: Retrieve-Common

        To access a small selection of records in the first cluster
         category of back to back record classes  and return those
         records that share common INTTONExx attributes within each
         respective record class.

        Assign the appropriate template and descriptor names.

        The descriptor values from  Request1 are used for the first retrieve
         and the upper range value is multiplied by the record factor
         as the value for the second retrieve.

        For the workload:
        The workload for the first retrieve is the same as Request1.
        The workload for the second retrieve is calculated by determining
         the number of clusters identified in cluster category 1 and
         determining the appropriate number of records accessed in that
         category.
        The result record set is the same as the number of records relevant
         by the source retrieve.

end procedure.
```

```
Request13()

        OBJECTIVE: Retrieve-Common

        For each retrieve, access all the records from the record class
         such that only 1/2 the records are relevant to that retrieve.
        Retrieve those records that share common INTTONExx attributes
         within each respective record class.

        Assign the appropriate template values.

        Assign the value "Onehalf" to the STRING010 attribute.

        For the workload:
        All cluster categories are examined by both the source and
         target retrieve. Make the appropriate assignments.
        All records are accessed by both the source and
         target retrieve. Make the appropriate assignments.
        The number of records relevant is half of the total number of
         records for each given record class.
        The result record set is the same as the number of records relevant
         by the source retrieve.

end procedure.
```

```
Request14()

        OBJECTIVE: Retrieve-Common


        For the source  retrieve, access all the records from the record
         class such that only 1/2 the records are relevant to that retrieve.
        For the target retrieve, access 1/16 the records of the second
         record class.
        Retrieve those records that share common INTTONExx attributes
         within each respective record class.

        Assign the appropriate template and descriptor names.

        Determine half the number of records for the first retrieve and
         assign this value to the INTONExx descriptor attribute for
         the first retrieve and also as the upper range value  INTONExx
         descriptor in the second retrieve.
        Determine 1/8 of the number of records for the first retrieve and
         subtract this value from the (half the number of records) value.
         This new value is the lower range  value for the INTONExx
         descriptor attribute of the target retrieve.


        For the workload:
         For the first retrieve:
          Identify the target cluster category and sum up all the clus-
          ters in this and all preceding categories for the total number
          of clusters examined. Do the same for all the records accessed.
          The number of records relevant to the source is exactly 1/2 the
          total number of records.
         For the second retrieve:
          Identify the target cluster category. The total number of
          clusters in this category is the number of clusters examined.
          Do the same for the number of records accessed by the target
          request.  The number of records relevant to the target request
          is the difference between its INTONExx descriptor values.
        The result record set is the same as the number of records
          relevant by the target retrieve.

end procedure;
```

Procedure Gen_reqset5()

```
    print table headings for the report file and workload file

    perform Request15();
    perform Request16();

end procedure;
```

Request15()

```
    OBJECTIVE: Insert (a record into an existing cluster)

    Assign the appropriate template and descriptor names.

    Determine the number of backends in the system: the value for
     the multiple attribute is one greater than the number of backends.
    Access the number of attributes per record class and write the
     appropriate number of string values, Xxxxxxxxx.

    For the workload:
    No clusters are examined and the database is not accessed.
    One record is inserted.

end procedure;
```

Request16()

```
    OBJECTIVE: Insert (a record into a new cluster)

    Assign the appropriate template and descriptor names.

    To determine the new value for the descriptor attribute:
     Determine the number of records in the first cluster category.
     Multiply this value by 100, then divide by the number of clusters
     in the first cluster category. The result is the new value for
     the descriptor attribute.
    The multiple attribute's value is "One"
    Access the number of attributes per record class and write the
     appropriate number of string values, Xxxxxxxxx.

    For the workload:
    No clusters are examined and the database is not accessed.
    One record is inserted.

end procedure;
```

Procedure Gen_reqset6()

```
        print table headings for the report file and workload file

        perform Request17();
        perform Request18();
        perform Request19();
        perform Request20();
        perform Request21();
        perform Request22();
        perform Request23();
        perform Request24();

end procedure;
```

Request17()

```
        OBJECTIVE: Delete

        Assign the appropriate template and descriptor names.

        The descriptor attribute values are the same as Request 1.

        For the workload:
         The workload is the same as request 1.

end procedure;
```

Request18()

```
        OBJECTIVE: Delete

        Request 18 has been stubbed.

end procedure;
```

Request19()

```
        OBJECTIVE: Delete

        Assign the appropriate template and descriptor names.

        The descriptor attribute value is calculated as follows:
        Subtract 25% of the total records from the total records value.
        The result is the descriptor value.

        For the workload:
        To determine the volume of the database deleted:
         Sum all the records between the descriptor attribute value
```

```
        and the total records  Divide this sum by the total records
        value.
       To determine the volume of the database accessed:
        Identify the cluster that stores the target descriptor value.
        Sum all the records in this cluster plus all the records in those
        clusters that follow up through the last cluster category. Divide
        the sum by the total number of records.

end procedure;
```

Request20()

```
        OBJECTIVE: Delete

        Assign the appropriate template and descriptor names.

        The descriptor attribute values are the same as Request 7.
        The string assignment is the same as Request 7.

        For the workload:
         The workload is the same as request 7.

end procedure:
```

Request21()

```
        OBJECTIVE: Delete

        Assign the appropriate template name.

        The string assignment is the same as Request 8.

        For the workload:
         The workload is the same as request 8.

end procedure;
```

Request22()

```
        OBJECTIVE: Delete

        Assign the appropriate template name.

        The string assignment is the same as Request 9.

        For the workload:
         The workload is the same as request 9.

end procedure;
```

Request23()

OBJECTIVE: Delete

Assign the appropriate template name.

The string assignment is the same as Request 10.

For the workload:
The workload is the same as request 10.

end procedure:

Request24()

OBJECTIVE: Delete

Assign the appropriate template and descriptor names.

The descriptor attribute value is the same as Request 11.
The string assignment is the same as Request 11.

For the workload:
The workload is the same as request 11.

end procedure:

```
procedure Format_test_benchmark_dbs ( input: num_be, num_config,
                                      output: lg_db_set, med_db_set,
                                          small_db_set);

/* This procedure generates the test_benchmark_databases          */
/* for the report                                                 */

        for each database size,
        write a separate database set file (large, med, small) such that
        db_sz[i], i ranging from 0 to 2 do

        /* The following code will generate the response time     */
        /* reduction configurations                               */

                for b_ends_in_use = 1 to num_be do
                        config_num = num_be

                        write config_num

                        mbytes_per_be = (db_sz[i]/4) / num_be

                        /* dividing the db_size[i] by 4 distributes */
                        /* the database evenly to each of the record*/
                        /* classes                                  */

                        write b_ends_in_use

                        for each rcd_sz[p], p ranging from 0 to 3  do

                                num_rcd_per_be =mbytes_per_be / rcd_sz[p

                                write  rcd_sz[p],
                                write  num_rcd_per_be, mbytes_per_be

                        end do

                        write db_sz[i]

                end do
```

```
        * The following code will generate the response time      *
        * invariance configurations                                *

                config_num = config_num - 1;
                b_ends_in_use  = 2;

              while b_ends_in_use is less than or equal to num_be do

                      write config_num, b_ends_in_use

                      mbytes_per_be = db_sz[i]/4

                      db_sz_in_use = (mbytes_per_be) * (b_ends_in_use)

                      for each rcd_sz[p], p ranging from 0 to 3  do

                              num_rcd_per_be =mbytes_per_be / rcd_sz[p]

                              write  rcd_sz[p], num_rcd_per_be,
                              write  mbytes_per_be

                      end do

                      write db_sz_in_use

                      b_ends_in_use = b_ends_in_use + 1

                      config_num = config_num + 1

              end while
      end for;
end procedure;

/* b_ends_in_use is the number of backends used for a specific configuration.*/
/* config_num is a specific configuration (number).                          */
/* rcd_sz is one of four record sizes from four_rcd_sz[]                     */
/* num_rcd_per_be is the number of records per backend (per record class)    */
/* db_sz_in_use is the database size created for the invariance configur-    */
/* ations                                                                    */
/* mbytes_per_be is Mbytes per backend (per record class).                   */
```

APPENDIX B: CAD GENERATED REPORTS

The reports contained in this appendix comprise only those generated for the small database set. The CAD system generates similar reports for the large and medium database sets as well. The reports are based upon a system defined to have the following:

maximum disk storage:	375	Megabytes
available disk storage:	300	Megabytes
disk track size:	4000	bytes
number of backends:	3	

The first set of reports are generated by the Test-Database Component. The second set of reports are generated by the Test-Transaction Mix Component.

THE RECORDS-PER-BLOCK RELATIONSHIP

Record Sizes in Bytes	Records per Block
2000	2
1000	4
400	10
200	20

THE NUMBER OF RECORDS PER CLUSTER CATEGORY TABLE

Blocks per Cluster:	Record Size in Bytes:			
	2000	1000	400	200
2	4	8	20	40
3	6	12	30	60
4	8	16	40	80
5	10	20	50	100
6	12	24	60	120
7	14	28	70	140
8	16	32	80	160
9	18	36	90	180
10	20	40	100	200

SMALL_DB_TEST_CONFIGURATIONS

Configuration Number	Number of Backends	Record Size in Bytes	Number of Records per Backend	Mybytes per Backend	Database Size in Mbytes
1	1	2000	9372	18744000	
		1000	18744	18744000	
		400	46860	18744000	
		200	93720	18744000	74.976
2	2	2000	4686	9372000	
		1000	9372	9372000	
		400	23430	9372000	
		200	46860	9372000	74.976
3	3	2000	3124	6248000	
		1000	6248	6248000	
		400	15620	6248000	
		200	31240	6248000	74.976
4	2	2000	9372	18744000	
		1000	18744	18744000	
		400	46860	18744000	
		200	93720	18744000	149.952
5	3	2000	9372	18744000	
		1000	18744	18744000	
		400	46860	18744000	
		200	93720	18744000	224.928

RECORD BLOCK DISTRIBUTION TABLE
SMALL DATABASE
THE NUMBER OF BACKENDS: 3

Record Size in Bytes	Number of Blocks per Cluster	Number of Records per Cluster	Total Number of Clusters	Total Number of Records	Total Number of Blocks
2000	2	4	86	344	172
	3	6	87	522	261
	4	8	87	696	348
	5	10	87	870	435
	6	12	87	1044	522
	7	14	87	1218	609
	8	16	87	1392	696
	9	18	87	1566	783
	10	20	86	1720	860
1000	2	8	86	688	172
	3	12	87	1044	261
	4	16	87	1392	348
	5	20	87	1740	435
	6	24	87	2088	522
	7	28	87	2436	609
	8	32	87	2784	696
	9	36	87	3132	783
	10	40	86	3440	860
400	2	20	86	1720	172
	3	30	87	2610	261
	4	40	87	3480	348
	5	50	87	4350	435
	6	60	87	5220	522
	7	70	87	6090	609
	8	80	87	6960	696
	9	90	87	7830	783
	10	100	86	8600	860
200	2	40	86	3440	172
	3	60	87	5220	261
	4	80	87	6960	348
	5	100	87	8700	435
	6	120	87	10440	522
	7	140	87	12180	609
	8	160	87	13920	696
	9	180	87	15660	783
	10	200	86	17200	860

THE RTR CONFIGURATION NUMBER 1

Number of Blocks / BE
172
261
348
435
522
609
696
783
860

THE RTR CONFIGURATION NUMBER 2

Number of Blocks / BE	
86	86
131	130
174	174
217	218
261	261
305	304
348	348
391	392
430	430

THE RTR CONFIGURATION NUMBER 3

Number of Blocks / BE		
58	57	57
87	87	87
116	116	116
145	145	145
174	174	174
203	203	203
232	232	232
261	261	261
286	287	287

SMALL DATABASE
LARGE RECORD CLASS
RECORD SIZE: 2000

REQUEST SET 1

Request Number:	RETRIEVAL Request Queries:
1	((TEMP=Templg)and(INTONELG>=109)and(INTONELG<=120))
3	((TEMP=Templg)and(INTTWOLG<=2343))

REQUEST SET 2

Request Number:	UPDATE Request Queries:
4	((TEMP=Templg)and(INTTWOLG<=1172))(STR000001 = Oneeighth)
5	((TEMP=Templg)and(INTTWOLG<=2343))(STR000005 = Onequartr)
6	((TEMP=Templg)and(INTTWOLG > 4686))(STR000010 = Onehalf)

REQUEST SET 3

Request Number:	RETRIEVAL Request Queries:
7	((TEMP=Templg)and(INTTWOLG <= 4686)and(STR000001 = Oneeighth))
8	((TEMP=Templg)and(STR000001 = Oneeighth))
9	((TEMP=Templg)and(STR000005 = Onequartr))
10	((TEMP=Templg)and(STR000010 = Onehalf))
11	((TEMP=Templg)and(INTTWOLG >= 4687)and(STR000010 = Onehalf))

REQUEST SET 4

Request Number:	RETRIEVE-COMMON Request Specifications:
12	RETRIEVE((TEMP=Templg)and(INTONELG>=109) and(INTONELG<=120) COMMON(INTONELG.INTONEMEDLG) RETRIEVE((TEMP=Tempmedlg)and(INTONEMEDLG<=240)) (INTONEMEDLG)
13	RETRIEVE((TEMP=Templg)and(STR000010=Onehalf) (INTTWOLG) COMMON(INTONELG.INTONEMEDLG) RETRIEVE((TEMP=Tempmedlg)and(STR000010=Onehalf) (INTTWOMEDLG)
14	RETRIEVE((TEMP=Templg)and(INTONELG<=4686)) (INTONELG) COMMON(INTTWOLG.INTTWOMEDLG) RETRIEVE((TEMP=Tempmedlg)and(INTONEMEDLG>=3515) and(INTONEMEDLG<=4686))(INTONEMEDLG)

REQUEST SET 5

Request Number:	INSERT Request Queries:
15	(<TEMPLATE.Templg>.<INTONELG,1>.<INTTWOLG,1>,<Multiple,Four>, <STR000001,Xxxxxxxxx>,....,<STR000196.Xxxxxxxxx>)
16	(<TEMPLATE.Templg>,<INTONELG,1>.<INTTWOLG,400>,<Multiple,One>, <STR000001,Xxxxxxxxx>,....,<STR000196.Xxxxxxxxx>)

REQUEST SET 6

Request Number:	DELETE Request Queries:
17	((TEMP=Templg)and(INTONELG>=109)and(INTONELG<=120))
19	((TEMPLATE = Templg)and(INTTWOLG>= 7030))
20	((TEMP=Templg)and(INTTWOLG <= 4686)and(STR000001 = Oneeighth))
21	((TEMP=Templg)and(STR000001 = Oneeighth))
22	((TEMP=Templg)and(STR000005 = Onequartr))
23	((TEMP=Templg)and(STR000010 = Onehalf))
24	((TEMP=Templg)and(INTTWOLG <= 4687)and(STR000010 = Onehalf))

SMALL DATABASE

LARGE RECORD CLASS

RECORD SIZE: 2000

REQUEST SET 1 WORKLOAD
RETRIEVE QUERIES

Request Number	Number of Clusters Examined	Volume of Database Accessed	Volume of Database Retrieved
1	86	344 records	12 records
3	339	25.0960 %	25.0000 %

REQUEST SET 2 WORKLOAD
UPDATE QUERIES

Request Number	Number of Clusters Examined	Volume of Database Accessed	Volume of Database Transacted
4	212	12.5694 %	12.5053 %
5	339	25.0960 %	25.0000 %
6	261	50.0640 %	50.0000 %

REQUEST SET 3 WORKLOAD
RETRIEVAL QUERIES

Request Number	Number of Clusters Examined	Volume of Database Accessed	Volume of Database Retrieved
7	521	50.0854 %	12.5000 %
8	781	100.00 %	12.51 %
9	781	100.00 %	25.00 %
10	781	100.00 %	50.00 %
11	261	50.0640 %	50.0107 %

REQUEST SET 4 WORKLOAD
RETRIEVAL QUERIES

Request Number	Number of Clusters Examined by the Source Request	Number of Records Accessed by the Source Request	Number of Records Relevant to the Source Request	Number of Clusters Examined by the Target Request	Number of Records Accessed by the Target Request	Number of Records Relevant to the Target Request	Size of the Result Record Set in Records
12	86	344	12	86	688	240	12
13	781	9372	4686	781	18744	9372	4686
14	521	4694	4686	87	1740	1171	1171

REQUEST SET 5 WORKLOAD
INSERT QUERIES

Request Number	Number of Clusters Examined	Volume of Database Accessed	Volume of Database Inserted
15	-	-	1 record
16	-	-	1 record

REQUEST SET 6 WORKLOAD
DELETE QUERIES

Request Number	Number of Clusters Examined	Volume of Database Accessed	Volume of Database Deleted
17	86	344 records	12 records
19	121	25.07 %	24.99 %
20	521	50.0854 %	12.5000 %
21	781	100.00 %	12.51 %
22	781	100.00 %	25.00 %
23	781	100.00 %	50.00 %
24	261	50.0640 %	50.0107 %

LIST OF REFERENCES

1. Demurjian, S.A., Hsiao D.K., Menon M.J., "A Multi-Backend Database System for Performance Gains, Capacity Growth and Hardware Upgrade", *Proceedings of the Second International Conference on Data Engineering*, 1986.
2. Vincent.J.R. *A Performance Measurement Methodology for a Multi-Backend Database System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1985.
3. Kovalchik, J. G., *Performance Evaluation Tools for a Multi-Backend Database System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1983.
4. Tekampe, R. C., and Watson, R. J., *Internal and External Performance Measurement Methodologies for Database Systems*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1984.
5. Strawser, P. R., *A Methodology for Benchmarking Relational Database Machines*, Ph.D. Dissertation, The Ohio State University, Columbus, Ohio, March 1984.
6. Hawthorn, P. B., and Stonebraker, M., "Performance Analysis of a Relational Data Base Management System," *Proceedings of the ACM SIGMOD Conference on Management of Data*, Boston, May 30-June 1, 1979.
7. The Ohio State University, Columbus, Ohio, Technical Report, OSU-CISRC-TR-81-7, *Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)*, by D. K. Hsiao and M. J. Menon, July 1981.
8. The Ohio State University, Columbus, Ohio, Technical Report, OSU-CISRC-TR-81-8, *Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part II)*, by D. K. Hsiao and M. J. Menon, July 1981.
9. Hsiao, D. K., and Harary, F., "A Formal System for Information Retrieval from Files," *Communications of the ACM*, Vol. 13, No. 2, February 1970: Corrigenda, Vol 13., No. 4, April 1970.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5000	2
2.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
3.	Curriculum Officer, Code 37 Computer Technology Programs Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor David K. Hsiao, Code 52Hq Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
5.	Steven A. Demurjian, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
6.	CDR. J.Anand, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
7.	MAJOR George P. Fenton Development Center, C3 MCDEC Quantico, Virginia 22134	5
8.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2

219519

Thesis
F254846
c.1

Fenton
A computer aided design for the generation of test transactions and test databases and for the benchmarking of parallel, Multiple Backend Database Systems.